VeriCall Edge™ V2.2

Function Reference Manual

Revision 2.21 February 2006

Copyright and Trademarks

© Copyright 2006 Trinity Convergence, Inc. All rights reserved.

VeriCall® is a Registered Trademark and VeriCall Edge™ is a Trademark of Trinity Convergence, Inc. All other products or services referenced may be trademarks or service marks of their respective companies or organizations.

Specific versions of VeriCall Edge contain software that includes the RSA Data Security, Inc. MD5 Message-Digest Algorithm.

Trinity Convergence Confidential

This manual contains material that is considered Trinity Convergence Confidential and is covered under the terms and conditions of the Trinity Convergence Non-disclosure Agreement.

Notice and Manual Information

Reasonable efforts have been made to ensure the accuracy of this document. However, Trinity Convergence, Inc. assumes no liability resulting from any omissions in this document, or from the use of the information obtained. Trinity Convergence, Inc reserves the right to update or change this document without obligation to notify any person of such updates or changes.

Electronic versions of this manual may be used/downloaded for personal use, or referenced in another document. The content itself may not be published commercially in print or electronic form, edited, translated, or otherwise altered without the permission of Trinity Convergence, Inc.

VeriCall Edge Function Reference Manual
Manual Number MAN20002
Manual Revision 2.21
February 2006

Contact Information

Trinity Convergence, Inc. is based in Research Triangle Park, North Carolina, USA and in Cambridge, UK. If you need Sales or Technical Support information, please contact one of the main offices listed below or refer to our web site.

Corporate Headquarters (Research Triangle Park, USA)

Trinity Convergence
2525 Meridian Pkwy
Suite 240
Durham, NC 27713
USA

Tel: 919-433-7000
Fax: 919-433-7001

European Headquarters (Cambridge, UK)

Trinity Convergence
Cambridge Business Park
Cowley Road
Cambridge
CB4 0WZ, UK

Tel: +44 (0) 1223-435550
Fax: +44 (0) 1223-435560

Internet/Email

Internet: www.trinityconvergence.com

Email: support@trinityconvergence.com

Table of Contents

Chapter/Section **Page**

Chapter 1 Introduction....1
1.1 Using This Manual....1
1.1.1 Contents and Structure....1
1.1.2 Conventions....2
1.2 Related Publications....3

Chapter 2 Introduction to VeriCall Edge™....5
2.1 Introducing the VeriCall Edge System....5
2.2 Open Architecture....5
2.2.1 Control Interface....6
2.2.2 Media Interface....6
2.2.3 Packet Interface....6

Chapter 3 VeriCall Edge™ System API....7
3.1 VeriCall Edge System API Function Descriptions....7
3.2 VeriCall Edge System API Functions....8

Chapter 4 VeriCall Edge™ User API....39
4.1 VeriCall Edge User API Function Descriptions....39
4.2 VeriCall Edge User API Functions....40

Chapter 5 VeriCall Edge™ Media Control API....49
5.1 VeriCall Edge Media Control API Function Descriptions....49
5.2 VeriCall Edge Media Control Functions....50

Chapter 6 VeriCall Edge™ Data Manager API....77
6.1 VeriCall Edge Data Manager API Function Descriptions....77
6.2 VeriCall Edge Data Manager Functions....78

Chapter 7 VeriCall Edge™ SIP API....103

7.1 VeriCall Edge SIP API Function Descriptions 103
7.2 VeriCall Edge SIP API Functions 104

Chapter 8 VeriCall Edge™ H.323 API 181
8.1 VeriCall Edge H.323 API Function Descriptions 181
8.2 VeriCall Edge H.323 API Functions 182

Chapter 9 VeriCall Edge™ NAT API 225
9.1 VeriCall Edge NAT API Function Descriptions 225
9.2 VeriCall Edge NAT API Functions 226

Appendix Page

Appendix A Return/Error Codes 245
A.1 Return/Error Codes 245

Appendix B Structures and Enumerations 253
B.1 t_ve_system_s 253
B.2 t_ve_init_config_s 254
B.3 t_ve_version_s 255
B.4 t_ve_system_logging_s 256
B.5 t_ve_last_stream_configuration_s 257
B.6 t_ve_run_vericall_edge_config_s 258
B.6.1 t_ve_audio_video_module_state_s 259
B.6.2 t_ve_audio_module_state_s 259
B.7 t_ve_ip_default_configuration_s 260
B.7.1 t_ve_ip_default_attributes_s 261
B.8 t_ve_ip_settings_default_configuration_s 261
B.9 t_ve_ajbm_algorithm_default_configuration_s 262
B.9.1 t_ve_ajbm_algorithm_default_lip_sync_config_s 263
B.10 t_ve_line_ecan_configuration_s 264
B.10.1.1 t_ve_line_ecan_16ms_ecan_configuration_s 265
B.11 t_ve_acoustic_ecan_configuration_s 266
B.11.1.1 t_ve_acoustic_ecan_aec_configuration_s 267
B.11.1.2 t_ve_acoustic_ecan_aec_complex_configuration_s 268
B.11.1.3 t_ve_acoustic_ecan_aes_configuration_s 271
B.12 t_ve_gain_control_configuration_s 273
B.13 t_ve_ntd_algorithm_default_configuration_s 274
B.14 t_ve_ntg_algorithm_default_configuration_s 275
B.15 t_ve_pre_proc_default_configuration_s 275
B.15.1 t_ve_null_pre_proc_default_configuration_s 276
B.15.2 t_ve_g711_pre_proc_default_configuration_s 277
B.15.3 t_ve_video_pre_proc_default_configuration_s 277
B.16 t_ve_post_proc_default_configuration_s 280
B.16.1 t_ve_null_post_proc_default_configuration_s 281
B.16.2 t_ve_g711_post_proc_default_configuration_s 281
B.16.3 t_ve_video_post_proc_default_configuration_s 282
B.17 t_ve_sip_default_configuration_s 284
B.18 t_ve_tx_lip_sync_default_configuration_s 287
B.19 t_ve_data_end_point_default_configuration_s 287

B.20 t_ve_video_tear_protect_default_configuration_s....288
B.21 t_ve_channel_s....289
B.21.1 t_ve_channel_3_way_calling_config_s....290
B.21.2 t_ve_channel_data_configuration_s....291
B.21.2.1 t_ve_channel_pre_conditioning_s....292
B.21.2.2 t_ve_channel_post_conditioning_s....292
B.21.3 t_ve_channel_packet_configuration_s....293
B.21.3.1 t_ve_channel_port_configuration_s....294
B.22 t_ve_callback_channel_data_configuration_u....295
B.22.1 t_ve_callback_audio_data_channel_config_s....296
B.22.2 t_ve_callback_video_data_channel_config_s....296
B.23 t_ve_callback_network_open_s....297
B.24 t_ve_stream_configuration_s....298
B.25 t_ve_user_handle_s....299
B.26 t_ve_user_channel_details_s....300
B.26.1 t_ve_user_channel_association_s....301
B.26.1.1 t_ve_user_channel_selection_s....301
B.27 t_ve_media_description_s....302
B.27.1 t_ve_media_field_s....303
B.27.2 t_ve_media_ip_configuration_s....304
B.27.3 t_ve_media_attrib_mapping_s....305
B.27.3.1 t_ve_media_rtp_attributes_s....306
B.28 t_ve_media_rtp_configuration_u....307
B.28.1 t_ve_media_g711_configuration_s....308
B.28.2 t_ve_media_g729a_configuration_s....308
B.28.3 t_ve_media_g723_1_configuration_s....309
B.28.4 t_ve_media_g726_configuration_s....309
B.28.5 t_ve_media_gsm_amr_configuration_s....310
B.28.6 t_ve_media_ilbc_configuration_s....312
B.28.7 t_ve_media_h263_configuration_s....313
B.28.8 t_ve_media_mpeg4_configuration_s....314
B.29 t_ve_media_definition_s....315
B.29.1 t_ve_media_g711_vad_encoder_algorithm_configuration_s....317
B.29.2 t_ve_media_g711_vad_decoder_algorithm_configuration_s....317
B.29.3 t_ve_media_g729a_encoder_algorithm_configuration_s....318
B.29.4 t_ve_media_g729a_decoder_algorithm_configuration_s....318
B.29.5 t_ve_media_g723_1_encoder_algorithm_configuration_s....319
B.29.6 t_ve_media_g723_1_decoder_algorithm_configuration_s....319
B.29.7 t_ve_media_g726_encoder_algorithm_configuration_s....320

B.29.8 t_ve_media_g726_decoder_algorithm_configuration_s....320
B.29.9 t_ve_media_gsm_amr_encoder_algorithm_configuration_s....321
B.29.10 t_ve_media_gsm_amr_decoder_algorithm_configuration_s....323
B.29.11 t_ve_media_cn_encoder_algorithm_configuration_s....324
B.29.12 t_ve_media_cn_decoder_algorithm_configuration_s....324
B.29.13 t_ve_media_te_algorithm_configuration_s....325
B.29.14 t_ve_media_video_encoder_h263_algorithm_configuration_s....325
B.29.15 t_ve_media_video_decoder_h263_algorithm_configuration_s....326
B.29.16 t_ve_media_video_encoder_mpeg4_algorithm_configuration_s....327
B.29.16.1 t_ve_media_video_user_data_s....328
B.29.16.2 t_ve_media_video_pixel_aspect_ratio_s....329
B.29.16.3 t_ve_media_video_stream_type_info_s....330
B.29.16.4 t_ve_media_video_cyclic_intra_refresh_s....331
B.29.16.5 t_ve_media_video_pid_rate_control_s....332
B.29.16.6 t_ve_media_video_buffer_verifier_params_s....333
B.29.16.7 t_ve_media_video_intra_control_s....334
B.29.16.8 t_ve_media_video_compression_params_s....335
B.29.17 t_ve_media_video_decoder_mpeg4_algorithm_configuration_s....336
B.29.18 t_ve_media_video_frame_type_selection_s....336
B.29.19 t_ve_media_eg711_encoder_algorithm_configuration_s....337
B.29.20 t_ve_media_eg711_decoder_algorithm_configuration_s....338
B.29.21 t_ve_media_ilbc_encoder_algorithm_configuration_s....339
B.29.22 t_ve_media_ilbc_decoder_algorithm_configuration_s....340
B.29.23 t_ve_media_video_encoding_mode_rate_s....341
B.29.24 t_ve_media_video_mpeg4_stream_container_s....342
B.30 t_ve_network_control_u....342
B.31 t_ve_audio_control_u....343
B.31.1 t_ve_audio_control_play_start_s....344
B.31.2 t_ve_audio_control_record_start_s....344
B.31.3 t_ve_audio_control_lip_sync_adjustment_s....345
B.31.4 t_ve_audio_control_data_end_point_selector_s....345
B.31.5 t_ve_audio_control_data_audio_mute_s....346
B.32 t_ve_video_control_u....347
B.32.1 t_ve_video_control_scale_s....348
B.32.2 t_ve_video_control_mute_s....349
B.32.3 t_ve_video_control_move_and_resize_s....349
B.32.4 t_ve_video_control_image_encode_s....350
B.32.5 t_ve_video_control_lip_sync_adjustment_s....350
B.32.6 t_ve_video_control_data_end_point_selector_s....351

B.32.7 t_ve_video_control_reset_pre_proc_params_s....352
B.32.8 t_ve_video_control_reset_post_proc_params_s....354
B.33 t_ve_user_msg_s....356
B.33.1 t_ve_user_msg_nat_discover_response_s....360
B.33.2 t_ve_user_msg_nat_user_addresses_response_s....360
B.33.3 t_ve_user_msg_nat_request_failed_s....361
B.33.4 t_ve_user_msg_nat_keep_alive_event_s....362
B.33.5 t_ve_user_msg_sip_session_started_s....363
B.33.6 t_ve_user_msg_sip_session_updated_s....364
B.33.7 t_ve_user_msg_sip_session_stopped_s....364
B.33.8 t_ve_user_msg_sip_invite_rx_s....365
B.33.9 t_ve_user_msg_sip_reinvite_rx_s....366
B.33.10 t_ve_user_msg_sip_invite_response_rx_s....367
B.33.11 t_ve_user_msg_sip_update_rx_s....368
B.33.12 t_ve_user_msg_sip_update_response_rx_s....369
B.33.13 t_ve_user_msg_sip_reinvite_response_rx_s....370
B.33.14 t_ve_user_msg_sip_register_response_rx_s....371
B.33.15 t_ve_user_msg_sip_refer_rx_s....372
B.33.16 t_ve_user_msg_sip_refer_response_rx_s....373
B.33.17 t_ve_user_msg_sip_refer_notify_rx_s....374
B.33.18 t_ve_user_msg_sip_message_rx_s....375
B.33.19 t_ve_user_msg_sip_message_response_rx_s....376
B.33.20 t_ve_user_msg_sip_options_rx_s....377
B.33.21 t_ve_user_msg_sip_options_response_rx_s....378
B.33.22 t_ve_user_msg_sip_info_rx_s....379
B.33.23 t_ve_user_msg_sip_info_response_rx_s....380
B.33.24 t_ve_user_msg_sip_service_rx_s....381
B.33.25 t_ve_user_msg_sip_service_response_rx_s....382
B.33.26 t_ve_user_msg_sip_operation_failed_s....383
B.33.27 t_ve_user_msg_sip_presence_subscribe_rx_s....384
B.33.28 t_ve_user_msg_sip_presence_notify_rx_s....385
B.33.29 t_ve_user_msg_sip_presence_subscribe_response_rx_s....386
B.33.30 t_ve_user_msg_sip_mw_notify_rx_s....387
B.33.31 t_ve_user_msg_sip_mw_notify_response_rx_s....388
B.33.32 t_ve_user_msg_sip_prack_rx_s....389
B.33.32.1 t_ve_sip_call_id_s....390
B.33.33 t_ve_user_msg_sip_dialog_subscribe_rx_s....391
B.33.34 t_ve_user_msg_sip_dialog_notify_rx_s....392
B.33.35 t_ve_user_msg_sip_dialog_subscribe_response_rx_s....393

B.33.36 t_ve_user_msg_h323_registration_response_s 394
B.33.37 t_ve_user_msg_h323_call_started_s 395
B.33.38 t_ve_user_msg_h323_call_stopped_s 396
B.33.39 t_ve_user_msg_h323_call_updated_s 396
B.33.40 t_ve_user_msg_h323_call_request_s 397
B.33.41 t_ve_user_msg_h323_call_response_s 397
B.33.42 t_ve_user_msg_h323_transfer_response_s 398
B.33.43 t_ve_user_msg_h323_call_forwarded_s 399
B.33.44 t_ve_user_msg_h323_forward_response_s 399
B.33.45 t_ve_user_msg_h323_user_input_received_s 400
B.33.46 t_ve_user_msg_h323_call_hold_response_s 401
B.33.47 t_ve_user_msg_h323_mwi_response_s 401
B.33.48 t_ve_user_msg_h323_message_waiting_s 402
B.33.49 t_ve_user_msg_h323_request_failed_s 403
B.33.50 t_ve_user_msg_h323_video_feature_request_s 404
B.33.51 t_ve_user_msg_detect_tone_dtmf_s 404
B.33.52 t_ve_user_msg_detect_tone_cp_s 405
B.33.53 t_ve_user_msg_detect_tone_2100_s 405
B.33.54 t_ve_user_msg_detect_tone_1100_s 406
B.33.55 t_ve_user_msg_received_rfc2833_digit_s 406
B.33.56 t_ve_user_msg_play_prompt_event_s 407
B.33.57 t_ve_user_msg_record_prompt_event_s 407
B.33.58 t_ve_user_msg_encode_image_event_s 408
B.33.59 t_ve_user_msg_video_encode_path_config_err_s 409
B.33.60 t_ve_user_msg_stream_event_s 411
B.33.61 t_ve_user_msg_module_state_transition_event_s 412
B.33.62 t_ve_user_msg_alarm_s 413
B.33.62.1 t_ve_alarm_subtype_ajbm_discard_buffer_full_s 414
B.33.62.2 t_ve_alarm_subtype_ajbm_extended_break_in_stream_s 414
B.33.62.3 t_ve_alarm_subtype_ajbm_underrun_s 415
B.33.62.4 t_ve_alarm_subtype_sip_alarm_s 416
B.33.62.5 t_ve_alarm_subtype_h323_alarm_s 416
B.33.63 t_ve_user_msg_shutdown_complete_s 417
B.33.64 t_ve_user_msg_network_connection_event_s 417
B.33.65 t_ve_user_msg_net_lookup_response_s 418
B.33.65.1 t_ve_net_lookup_record_s 419
B.33.65.2 t_ve_net_lookup_record_ipv4_address_s 420
B.33.65.3 t_ve_net_lookup_record_server_s 420
B.33.65.4 t_ve_net_lookup_naptr_s 421

B.33.66 t_ve_user_msg_net_request_failed_s 421
B.33.67 t_ve_user_msg_undecodeable_stream_s 422
B.33.68 t_ve_user_msg_decode_media_change_event_s 423
B.34 t_ve_callback_data_channel_descriptor_s 424
B.35 t_ve_callback_host_descriptor_s 425
B.36 t_ve_callback_device_query_u 426
B.36.1 t_ve_callback_audio_device_query_s 427
B.36.2 t_ve_callback_video_display_device_query_s 428
B.36.3 t_ve_callback_video_capture_device_query_s 429
B.37 t_ve_callback_network_config_s 430
B.38 t_ve_statistic_u 431
B.38.1 t_ve_stat_system_calls_s 432
B.38.2 t_ve_stat_channel_av_sync_s 432
B.38.2.1 t_ve_stat_av_sync_s 433
B.38.3 t_ve_stat_channel_video_encoder_s 433
B.38.3.1 t_ve_stat_video_encoder_s 434
B.38.4 t_ve_stat_channel_video_decoder_s 434
B.38.4.1 t_ve_stat_video_decoder_s 435
B.38.5 t_ve_stat_channel_rtcp_s 435
B.38.5.1 t_ve_stat_rtcp_s 436
B.38.6 t_ve_stat_channel_rtcp_peer_s 438
B.38.6.1 t_ve_stat_video_encoder_s 439
B.39 t_ve_sip_user_registration_s 440
B.39.1 t_ve_sip_user_uri_s 440
B.39.2 t_ve_sip_contact_info_s 441
B.39.3 t_ve_sip_registration_server_s 442
B.40 t_ve_sip_stack_liberalisation_s 443
B.41 t_ve_sip_stack_feature_support_s 443
B.42 t_ve_sip_routing_s 445
B.42.1 t_ve_sip_routing_data_s 445
B.43 t_ve_sip_date_s 447
B.44 t_ve_sip_tx_session_response_data_s 448
B.45 t_ve_sip_rx_session_response_data_s 449
B.45.1 t_ve_sip_authentication_data_s 450
B.45.2 t_ve_sip_content_data_s 450
B.46 t_ve_sip_presence_data_s 452
B.46.1 t_ve_sip_presence_xml_pidf_presence_s 453
B.46.1.1 t_ve_sip_presence_xml_pidf_tuple_s 454
B.46.1.2 t_ve_sip_presence_xml_pidf_status_s 455

B.46.1.3 t_ve_sip_presence_xml_pidf_contact_s 456
B.46.1.4 t_ve_sip_presence_xml_pidf_note_s 456
B.47 t_ve_sip_mw_data_s....457
B.47.1 t_ve_sip_mw_msg_summary_data_s 458
B.48 t_ve_sip_dialog_data_s....459
B.48.1 t_ve_sip_dialog_s 460
B.48.1.1 t_ve_sip_dialog_id_s 460
B.48.1.2 t_ve_sip_dialog_state_s 461
B.48.1.3 t_ve_sip_dialog_direction_s 462
B.48.1.4 t_ve_sip_dialog_duration_s 462
B.49 t_ve_sip_session_handle_s 463
B.50 t_ve_sip_privacy_identity_s 464
B.50.1 t_ve_sip_privacy_s 465
B.51 t_ve_remote_user_uri_s....466
B.52 t_ve_h323_user_registration_s 467
B.52.1 t_ve_h323_user_address_s 467
B.52.2 t_ve_h323_contact_info_s....468
B.53 t_ve_h323_msg_center_id_s....468
B.54 t_ve_h323_dest_address_s....469
B.55 t_ve_h323_call_handle_s....469
B.56 t_ve_h323_call_update_user_channel_details_s 470
B.56.1 t_ve_h323_call_update_user_channel_associations_s 471
B.56.2 t_ve_h323_user_channel_selection_s....472
B.56.3 t_ve_h323_call_update_attributes_s....473
B.57 t_ve_h323_name_id_s 474
B.58 t_ve_h323_default_configuration_s....475
B.58.1 t_ve_h323_system_configuration_s 476
B.58.2 t_ve_h323_ras_configuration_s....477
B.58.2.1 t_ve_h323_gatekeeper_s....478
B.58.3 t_ve_h323_q931_configuration_s 479
B.58.3.1 t_ve_h323_q931_annex_e_configuration_s 481
B.58.4 t_ve_h323_h245_configuration_s 482
B.58.5 t_ve_h323_h450_configuration_s 483
B.58.5.1 t_ve_h323_h450_2_configuration_s....484
B.58.5.2 t_ve_h323_h450_3_configuration_s....485
B.58.5.3 t_ve_h323_h450_4_configuration_s....486
B.58.5.4 t_ve_h323_h450_6_configuration_s....486
B.58.5.5 t_ve_h323_h450_7_configuration_s....487
B.59 t_ve_tone_definition_s....488

B.59.1 t_ve_tone_subgroup_s 489
B.59.2 t_ve_tone_subtone_s 490
B.60 t_ve_nat_default_configuration_s 491
B.61 t_ve_nat_channel_addresses_s 492
B.61.1 t_ve_nat_media_addresses_s 493
B.62 t_ve_nat_init_s 494
B.62.1 t_ve_nat_stun_init_s 495
B.62.2 t_ve_nat_turn_init_s 495
B.63 t_ve_nat_configuration_s 496
B.63.1 t_ve_nat_sip_addresses_s 497
B.64 t_ve_network_connection_s 498
B.65 t_ve_net_default_configuration_s 499
B.66 t_ve_net_lookup_query_s 500
B.67 t_ve_net_lookup_ipv4_address_s 501
B.68 t_ve_net_lookup_server_s 501
B.69 t_ve_net_lookup_naptr_s 502
B.70 t_ve_net_lookup_naptr_enum_s 503
B.71 t_ve_attribute_type_e 504
B.72 t_ve_return_code_e 510
B.73 t_ve_verbose_level_e 513
B.74 t_ve_system_logging_destination_e 514
B.75 t_ve_system_logging_method_e 515
B.76 t_ve_signaling_type_e 515
B.77 t_ve_tone_transmission_media_type_e 516
B.78 t_ve_tone_type_e 517
B.79 t_ve_tone_direction_e 518
B.80 t_ve_country_e 519
B.81 t_ve_ip_default_configuration_e 520
B.82 t_ve_ip_port_reopen_e 520
B.83 t_ve_process_selector_default_configuration_e 521
B.84 t_ve_default_configuration_boolean_e 521
B.85 t_ve_g711_default_mode_e 522
B.86 t_ve_line_16ms_ecan_nlp_default_settings_e 523
B.87 t_ve_acoustic_ecan_id_e 524
B.88 t_ve_acoustic_ecan_aes_mode_e 525
B.89 t_ve_line_ecan_id_e 526
B.90 t_ve_ajbm_algorithm_default_lip_sync_state_e 527
B.91 t_ve_gain_control_mode_e 528
B.92 t_ve_gain_control_agc_mode_e 528

B.93 t_ve_ntd_country_type_default_configuration_e 529
B.94 t_ve_video_default_local_image_enable_e 530
B.95 t_ve_video_default_image_rotate_e 530
B.96 t_ve_video_default_image_mirror_e 531
B.97 t_ve_video_color_format_e 532
B.98 t_ve_video_filter_selection_e 532
B.99 t_ve_data_end_point_default_type_e 533
B.100 t_ve_channel_data_format_e 534
B.101 t_ve_channel_config_boolean_e 534
B.102 t_ve_channel_type_config_e 535
B.103 t_ve_high_pass_filter_state_e 535
B.104 t_ve_media_data_type_e 536
B.105 t_ve_media_transport_type_e 536
B.106 t_ve_media_type_e 537
B.107 t_ve_media_ip_configuration_e 537
B.108 t_ve_media_g711_mode_e 538
B.109 t_ve_media_vad_enable_boolean_e 538
B.110 t_ve_media_id_e 539
B.111 t_ve_media_eg711_codec_and_law_e 541
B.112 t_ve_media_video_frame_type_selection_e 542
B.113 t_ve_media_video_encoding_mode_e 542
B.114 t_ve_media_video_profile_selection_e 543
B.115 t_ve_media_video_encoding_mode_rate_e 543
B.116 t_ve_media_video_quantization_param_e 544
B.117 t_ve_media_video_config_state_e 546
B.118 t_ve_media_video_simple_profile_e 547
B.119 t_ve_media_video_profile_level_e 548
B.120 t_ve_media_video_stream_type_e 549
B.121 t_ve_media_vad_state_e 549
B.122 t_ve_media_g723_1_rate_e 550
B.123 t_ve_media_g723_1_mode_e 551
B.124 t_ve_media_g723_1_hpf_enable_boolean_e 551
B.125 t_ve_media_g723_1_pf_enable_boolean_e 552
B.126 t_ve_media_gsm_amr_rate_e 553
B.127 t_ve_media_gsm_amr_mode_e 554
B.128 t_ve_media_gsm_amr_vad_mode_e 555
B.129 t_ve_media_gsm_amr_vad_state_e 556
B.130 t_ve_media_stream_direction_e 557
B.131 t_ve_last_stream_state_e 558

B.132 t_ve_module_state_e 559
B.133 t_ve_module_class_e 559
B.134 t_ve_network_control_type_e 560
B.135 t_ve_network_connection_type_e 561
B.136 t_ve_network_connection_event_e 562
B.137 t_ve_net_lookup_type_e 563
B.138 t_ve_net_service_type_e 564
B.139 t_ve_net_service_protocol_e 565
B.140 t_ve_net_lookup_record_type_e 566
B.141 t_ve_net_request_type_e 566
B.142 t_ve_net_request_failed_reason_e 567
B.143 t_ve_audio_control_type_e 568
B.144 t_ve_audio_control_direction_e 570
B.145 t_ve_audio_control_record_data_format_e 570
B.146 t_ve_audio_control_play_data_format_e 571
B.147 t_ve_audio_control_data_audio_mute_state_e 572
B.148 t_ve_audio_control_data_audio_mute_direction_e 573
B.149 t_ve_video_control_type_e 574
B.150 t_ve_video_control_filter_selector_e 576
B.151 t_ve_video_control_select_display_state_e 576
B.152 t_ve_video_control_rotate_e 577
B.153 t_ve_video_control_mirror_e 577
B.154 t_ve_video_control_mute_mode_e 578
B.155 t_ve_video_control_image_freeze_state_e 579
B.156 t_ve_video_control_encode_format_e 579
B.157 t_ve_video_config_error_reason_e 580
B.158 t_ve_alarm_type_e 581
B.159 t_ve_alarm_system_subtype_e 581
B.160 t_ve_alarm_sip_subtype_e 582
B.161 t_ve_alarm_h323_subtype_e 583
B.162 t_ve_user_msg_type_e 584
B.163 t_ve_user_msg_stream_event_reason_e 589
B.164 t_ve_user_msg_stream_event_type_e 590
B.165 t_ve_user_msg_module_state_transition_reason_e 590
B.166 t_ve_sip_request_failed_reason_e 591
B.167 t_ve_user_sub_channel_select_e 593
B.168 t_ve_user_sub_channel_index_e 594
B.169 t_ve_user_callback_return_code_e 594
B.170 t_ve_callback_transport_type_e 595

B.171 t_ve_callback_protocol_e 595
B.172 t_ve_callback_direction_e 596
B.173 t_ve_callback_audio_endian_e 596
B.174 t_ve_callback_data_channel_media_format_e 597
B.175 t_ve_callback_video_color_format_e 598
B.176 t_ve_callback_duplex_capability_e 598
B.177 t_ve_callback_transfermode_e 599
B.178 t_ve_callback_video_frame_buffer_e 599
B.179 t_ve_stats_type_e 600
B.180 t_ve_stat_av_sync_correlate_e 601
B.181 t_ve_sip_response_code_e 602
B.182 t_ve_sip_symmetric_mode_default_configuration_e 604
B.183 t_ve_sip_state_flag_default_configuration_e 605
B.184 t_ve_sip_deregister_contacts_flag_e 606
B.185 t_ve_sip_require_prack_e 606
B.186 t_ve_sip_date_state_e 607
B.187 t_ve_sip_tx_session_response_content_e 607
B.188 t_ve_sip_authentication_required_e 608
B.189 t_ve_sip_rx_session_response_content_e 608
B.190 t_ve_sip_call_hold_method_e 609
B.191 t_ve_sip_call_hold_state_e 609
B.192 t_ve_sip_presence_xml_pidf_status_basic_type_e 610
B.193 t_ve_sip_request_type_e 611
B.194 t_ve_sip_subscribe_state_e 614
B.195 t_ve_sip_presence_type_e 615
B.196 t_ve_sip_mw_start_flag_e 616
B.197 t_ve_sip_mw_indicatorstatus_e 617
B.198 t_ve_sip_mw_message_context_e 618
B.199 t_ve_sip_dialog_field_state_e 619
B.200 t_ve_sip_dialog_state_e 620
B.201 t_ve_sip_dialog_state_event_e 621
B.202 t_ve_sip_dialog_direction_e 622
B.203 t_ve_sip_transport_protocol_e 623
B.204 t_ve_sip_privacy_e 624
B.205 t_ve_h323_call_forward_reason_e 625
B.206 t_ve_h323_call_response_e 626
B.207 t_ve_h323_reg_response_e 629
B.208 t_ve_h323_call_hold_response_e 632
B.209 t_ve_h323_mwi_response_e 633

B.210 t_ve_h323_transfer_response_e 634
B.211 t_ve_h323_forward_response_e 636
B.212 t_ve_h323_request_type_e 637
B.213 t_ve_h323_request_failed_reason_e 639
B.214 t_ve_h323_name_id_char_set_e 640
B.215 t_ve_h323_name_id_restriction_e 641
B.216 t_ve_h323_call_waiting_flag_e 642
B.217 t_ve_h323_q931_annex_e_protocol_preference_e 643
B.218 t_ve_h323_gatekeeper_discovery_e 644
B.219 t_ve_h323_video_feature_request_e 645
B.220 t_ve_h323_call_start_method_e 646
B.221 t_ve_h323_hold_method_e 647
B.222 t_ve_h323_user_sub_channel_select_e 648
B.223 t_ve_user_msg_play_prompt_event_e 648
B.224 t_ve_user_msg_record_prompt_event_e 649
B.225 t_ve_user_msg_encode_image_event_e 649
B.226 t_ve_encode_image_event_type_e 650
B.227 t_ve_nat_discover_method_e 650
B.228 t_ve_nat_state_e 651
B.229 t_ve_nat_types_e 652
B.230 t_ve_nat_request_type_e 653
B.231 t_ve_nat_request_failed_reason_e 654
B.232 t_ve_nat_keep_alive_event_e 655
B.233 t_ve_user_msg_undecodeable_stream_reason_e 656

Appendix C Attribute to Structure Association 659
C.1 Attribute/Structure Association 659
C.2 Attribute/Index Association 662

Appendix D Header Files & Type Definitions 665
D.1 Header Files 666
D.2 Type Definitions 668
D.3 Macro Definitions 668

Appendix E VeriCall Edge™ Terminology 669

Table Page

Table A.1: VeriCall Edge API Return/Error Codes 245
Table C.1: VeriCall Edge Attribute to Structure Association 659
Table C.2: VeriCall Edge Attribute/Index Structure Association 662

Chapter 1

Introduction

This manual describes the VeriCall Edge™ API functions that are used to configure and use the VeriCall Edge software system. Section 1.1.1 below describes the contents of this manual.

1.1 Using This Manual

This Function Reference Manual provides detailed descriptions of the VeriCall Edge V2.2 API functions. For general information on using the VeriCall Edge software, please refer to the VeriCall Edge User Manual (MAN20003).

1.1.1 Contents and Structure

The contents and structure of this manual are as follows:

- Chapter 2 on Page 5 — Provides an introductory overview of the VeriCall Edge software.
- Chapter 3 on Page 7 — Provides detailed descriptions of the VeriCall Edge System API functions.
- Chapter 4 on Page 39 — Provides detailed descriptions of the VeriCall Edge User API functions.
- Chapter 5 on Page 49 — Provides detailed descriptions of the VeriCall Edge Media Control API functions.
- Chapter 6 on Page 77 — Provides detailed descriptions of the VeriCall Edge Data Manager API Functions.
- Chapter 7 on Page 103 — Provides detailed descriptions of the VeriCall Edge SIP API functions.
- Chapter 8 on Page 181 — Provides detailed descriptions of the VeriCall Edge H.323 API functions.
- Chapter 9 on Page 225 — Provides detailed descriptions of the VeriCall Edge NAT Traversal Module API functions.
- Appendix A on Page 245 — Provides detailed descriptions of the return and error codes used by the VeriCall Edge API functions.

- Appendix B on Page 253 — Details the various structures used by the VeriCall Edge API functions.
- Appendix C on Page 659 — Details the association between the VeriCall Edge attribute enumeration types and defined public structures.
- Appendix D on Page 665 — Gives the VeriCall Edge API public header files and type definitions.
- Appendix E on Page 669 — Provides a list of commonly used terms from the VeriCall Edge documentation set.

1.1.2 Conventions

This manual uses the following conventions:

- Hexadecimal numbers are prefixed by '0x'; for example, 0x00400000.
- This manual uses the standard k/K unit convention. That is, lower case 'k' is 1000, and upper case 'K' is 1024. For example, 16 kbits/s is used to represent 16000 bits per second.
- Names of directories, files and functions are given in bold; for example, **ve_media.h**.
- `Courier` font is used for code and syntax.
- Within discussions of syntax, text in italics highlights arguments; for example,
  - `VE_RunVeriCallEdge( UWord32 applicationId, t_ve_system_s *system_attrib )`
- Hypertext links within the manual are highlighted in blue.
- Note:

Text associated with this symbol highlights useful or important information.

- Warning Note:

Text associated with this symbol is a warning to the user and indicates a situation that could cause damage to the product or cause personal injury. Please read each warning carefully.

1.2 Related Publications

For more information on the VeriCall Edge software, refer to the following publications:

- **VeriCall Edge User Manual** — Provides information on how the VeriCall Edge software works, plus example User Applications.
- **VeriCall Edge Release Notes** — Provides last minute information on the VeriCall Edge release.
- **VeriCall Application Notes** — A number of Application Notes are available for use with the VeriCall and VeriCall Edge software. These provide additional information on topics relating to the VeriCall and VeriCall Edge systems. For details of the current list of Application Notes available, please refer to the Trinity Convergence web site.

Chapter 2

Introduction to VeriCall Edge™

This chapter gives a brief introduction to the VeriCall Edge V2IP software. The reader is directed to the VeriCall Edge User Manual (MAN20003) for a more detailed introduction to the VeriCall Edge software.

2.1 Introducing the VeriCall Edge System

The VeriCall Edge system is a complete and integrated V2IP platform offering the requisite media processing, packet handling, and call control functionality required to build a V2IP enabled client device. With VeriCall Edge, developers can add V2IP functionality to build edge devices and handsets without the need for costly and power hungry V2IP ASICs or Digital Signal Processors (DSPs). The VeriCall Edge software uses the available computing power within processors and cores traditionally used in high volume, low cost, and low power client device designs to deliver a comprehensive software-only V2IP solution.

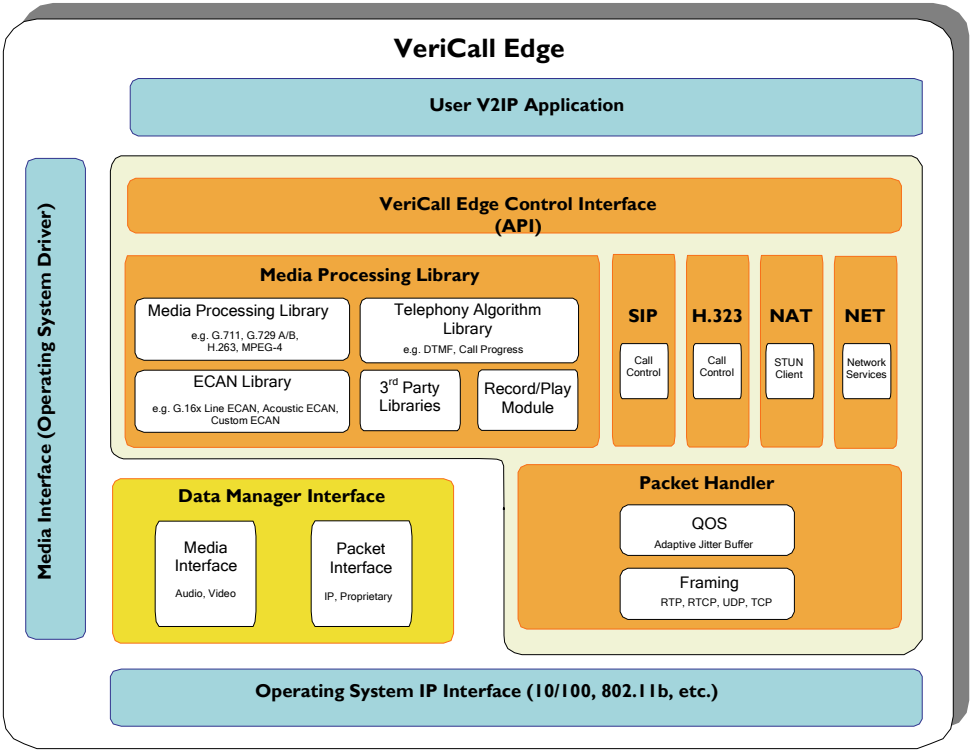
The VeriCall Edge system utilizes an open architecture, which means a software architecture whose interfaces are public. This includes industry standards as well as proprietary components whose specifications are made public. The open interfaces of the VeriCall Edge software allow OEMs to easily integrate custom or third party algorithms and protocols into their solution. It provides all the time to market benefits of a turnkey solution with the flexibility of an open architecture. The VeriCall Edge software allows vendors to easily V2IP enable their products while preserving their differentiation from competitors in the marketplace.

The VeriCall Edge software utilizes algorithms in the Media Processing Library to carry out the required media encoding/decoding, tone handling, acoustic and line echo cancellation, and other voice quality enhancement algorithms required for V2IP. Call control is made available using the standards compliant SIP and/or H.323 module or a 3rd party call control stack. This allows calls to be established or received from a rapidly increasing installed base of SIP compliant IP phones and H.323 gateways.

2.2 Open Architecture

The open architecture of the VeriCall Edge software is presented in Figure 1 on Page 6:

Figure 1 VeriCall Edge Open Architecture



2.2.1 Control Interface

The application control interface provides the VeriCall Edge public ANSI 'C' API. This interface, allows developers to rapidly implement a V2IP application utilizing the media processing and call control of the VeriCall Edge software.

2.2.2 Media Interface

This interface is used to transmit and receive channel media, for example G.711 audio data or H.263 video data from a physical audio device. Typical audio devices include headsets, handsets, Subscriber Line Interface Circuits (SLICs) or Digital Audio Adapters (DAAs).

2.2.3 Packet Interface

This interface is used to transmit and receive RTP framed data from the V2IP packet stream. The interface supports popular physical packet transports including Ethernet and 802.11x.

Chapter 3

VeriCall Edge™ System API

This chapter describes the VeriCall Edge System API functions. The VeriCall Edge API is used for VeriCall Edge system set-up and configuration as well as generating management information data and alarms.

3.1 VeriCall Edge System API Function Descriptions

For each function detailed in this section, the following are described (as appropriate):

- **Syntax** Command syntax.
- **Arguments** Description of the arguments.
- **Returns** Return codes from the function.
- **See Also** Other functions that perform related tasks.
- **Description** What the function can be used for.
- **Example** Short code sections showing how to use the function.

3.2 VeriCall Edge System API Functions

The VeriCall Edge System API functions are described on the following pages:

Function	Page
VE_RegisterApp	9
VE_MemoryResourceQuery	10
VE_Initialize	11
VE_SetSystemAttribute	12
VE_GetSystemAttribute	15
VE_UpdateSystemAttribute	17
VE_UpdateChannelAttribute	19
VE_RemoveSystemAttribute	21
VE_Commit	23
VE_RunVeriCallEdge	25
VE_RunVeriCallEdgeWithConfig	27
VE_Shutdown	31
VE_GetNextEvent	33
VE_GetVersion	34
VE_GetStatistics	35
VE_ClearStatistics	37

VE_RegisterApp

Syntax: `t_ve_return_code_e VE_RegisterApp( UWord32 *ve_application_id )`

Arguments:

ve_application_id	Pointer to area of local storage to hold the application's unique identifier.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_RegisterApp** enables each application that wishes to use the VeriCall Edge API to register itself with the VeriCall Edge system. The application ID is assigned and placed into the variable pointed to by *ve_application_id*. This application ID can then be used to verify the application's validity during subsequent VeriCall Edge API calls and for the routing of VeriCall Edge messages. The use of unique application identifiers allows multiple user applications simultaneous access to the VeriCall Edge API.

The first application to register is termed the "primary" application and is the only application allowed to call VE_Commit and VE_Shutdown.

Secondary applications can register with the VeriCall Edge system at any point after the primary application has registered.

Example: This example shows how to obtain an application identifier:

```
#include "ve_eapi.h"

/* Register the application to get an application ID */
{
   t_ve_return_code_e ret_code;
   UWord32 application_id;

   ret_code = VE_RegisterApp(&application_id);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_MemoryResourceQuery

Syntax:
```
t_ve_return_code_e  VE_MemoryResourceQuery(
                                    UWord32 *ve_resource_bytes_required )
```

Arguments:

ve_resource_bytes_required	Pointer to an area of memory to hold the returned number of bytes the VeriCall Edge system requires.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_Initialize**, **VE_Commit**

Description: **VE_MemoryResourceQuery** is an informational function used to return the context memory (heap) requirements of the VeriCall Edge system based on the current channel configurations, the attribute settings and the queue depth.

The size of the memory currently allocated or required, based on the current configuration, is returned to the area of storage pointed to by *ve_resource_bytes_required* and is measured in bytes.

Note that the context memory is not allocated until a call to VE_Commit is made, but context memory requirements are calculated when this function is called based on the parameters passed via the VE_Initialize function call, the default configuration and the current user configuration.

Example: This example shows how to query the amount of memory the VeriCall Edge system requires:

```
#include "ve_eapi.h"

/* Determine the dynamic memory requirements for VeriCall Edge */
{
   t_ve_return_code_e ret_code;
   UWord32 bytes_required

   ret_code = VE_MemoryResourceQuery(&bytes_required);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_Initialize

Syntax:

```
t_ve_return_code_e  VE_Initialize(
                      UWord32 ve_application_id,
                      t_ve_init_config_s *ve_initialize_config )
```

Arguments:

ve_application_id	The registered application ID.
ve_initialize_config	The initial configuration parameters of the VeriCall Edge system.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_SetSystemAttribute**, **VE_RemoveSystemAttribute**, **VE_GetSystemAttribute**, **VE_Commit.**

Description: **VE_Initialize** is used to initialize the VeriCall Edge system with the default configuration. The supplied argument, *ve_application_id*, is used to validate that the caller application has been registered with the system. The VeriCall Edge initialization parameters are passed in through the structure pointed to by *ve_initialize_config*. This structure is of type **t_ve_init_config_s** and is described in Appendix B.2 on Page 254.

If the initialization was successful, indicated by the return code VE_RETURN_CODE_SUCCESS, the function **VE_SetSystemAttribute** can be used to alter the default VeriCall Edge attribute values until the time of commit, **VE_Commit**.

Example: This example shows how to initialize the VeriCall Edge system:

```
#include "ve_eapi.h"

/* Initialize VeriCall Edge */
{
   t_ve_return_code_e ret_code;
   t_ve_init_config_s init_config;

   init_config.number_of_channels = MY_DEFS_NUMBER_OF_CHANNELS;
   init_config.number_of_users = MY_DEFS_NUMBER_OF_USERS;
   init_config.number_of_interfaces = MY_DEFS_NUMBER_OF_INTERFACES;

   ret_code = VE_Initialize(application_id, &init_config);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_SetSystemAttribute

Syntax:

```
t_ve_return_code_e  VE_SetSystemAttribute(
                        UWord32 ve_application_id,
                        t_ve_attribute_type_e ve_attribute_type,
                        UWord32 ve_attribute_index,
                        void *ve_user_data,
                        UWord32 size_of_data )
```

Arguments:

ve_application_id	The registered application ID.
ve_attribute_type	An enumerated value that is used for classification of the data entity.
ve_attribute_index	A unique key value that is assigned for each instance of a data entity.
ve_user_data	Pointer to the data structure to be stored.
size_of_data	Size of the passed data structure.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_GetSystemAttribute**, **VE_RemoveSystemAttribute**, **VE_Commit**, **VE_Initialize**

Description: **VE_SetSystemAttribute** allows the configuration attributes of the VeriCall Edge system to be altered. This function may only be used before the invocation of **VE_Commit**; any use of this function after such time will result in the return/error code VE_RETURN_CODE_ALREADY_COMMITTED.

The *ve_application_id* argument must be that of a previously registered application. The *ve_attribute_type* argument, used to identify the field to be set, must be one of the enumerated types defined in **t_ve_attribute_type_e**, as described in Appendix B.71 on Page 504. Associated VeriCall Edge structures to set these attributes are stated in Table C.1 on Page 659.

The *ve_attribute_index* argument is used to define an instance entry and should be unique for each instance. Associated attribute index values are stated in Table C.2 on Page 662.

Continued >>

The memory allocated in the pointer argument *ve_user_data*, must be of size *size_of_data* and be of equal size to the *ve_attribute_type* under consideration.

A macro (VE_SINGLETON_DATA_ENTITY) will be provided to define the *ve_attribute_index* argument for when there is only one instance of a data entity. The provided macro is defined in Appendix D.3 on Page 668.

Example: This example shows how to set a system attribute:

```
#include "ve_eapi.h"
#include <stdio.h>

/* Set the logging system attribute */
{
   t_ve_return_code_e ret_code;
   t_ve_system_logging_s system_logging;

   system_logging.destination =
                  VE_SYSTEM_LOGGING_DESTINATION_FILEHANDLE;
   system_logging.method = VE_SYSTEM_LOGGING_METHOD_DIRECT;
   system_logging.file_handle = stdout;
   system_logging.verbose = VE_VERBOSE_LEVEL_WARNING;

   ret_code = VE_SetSystemAttribute(application_id
              VE_ATTRIBUTE_TYPE_SYSTEM_LOGGING,
              VE_SINGLETON_DATA_ENTITY,
              &system_logging,
              sizeof(t_ve_system_logging_s));

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

Example: This example shows how to set a system attribute:

```
#include "ve_eapi.h"

/* Set the AEC configuration system attribute for a channel */
{
   t_ve_return_code_e ret_code;
   t_ve_acoustic_ecan_configuration_s ecan_config;
   t_ve_acoustic_ecan_aec_configuration_s *aec_config;

   ecan_config.ve_acoustic_ecan_id = AEC_ACOUSTIC_ECAN_ID;
   aec_config = &ecan_config.ve_acoustic_ecan_config.aec;
```

Continued >>

```
    aec_config->ecan_enable  = VE_DEFAULT_CONFIGURATION_TRUE;
    aec_config->configuration_profile = 0;
    aec_config->bulk_delay  = 50;
    aec_config->post_gain  = 256;
    aec_config->test_mode  = 0;
    aec_config->write_audio  = VE_DEFAULT_CONFIGURATION_FALSE;

    ret_code = VE_SetSystemAttribute(application_id
               VE_ATTRIBUTE_TYPE_DEFAULT_ACOUSTIC_ECAN,
               channel_number,
               &ecan_config,
               sizeof(t_ve_acoustic_ecan_configuration_s));

    if(ret_code != VE_RETURN_CODE_SUCCESS)
    {
       /* Handle error */
    }
}
```

VE_GetSystemAttribute

Syntax:

```
t_ve_return_code_e  VE_GetSystemAttribute(
                        UWord32 ve_application_id,
                        t_ve_attribute_type_e ve_attribute_type,
                        UWord32 ve_attribute_index,
                        void *ve_user_data,
                        UWord32 size_of_data )
```

Arguments:

ve_application_id	The registered application ID.
ve_attribute_type	An enumerated value, which is used for classification of the data entity.
ve_attribute_index	A unique key value that is assigned for each instance of a data entity.
ve_user_data	Pointer to the data structure to be stored.
size_of_data	Size of the passed data structure.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_SetSystemAttribute**, **VE_Initialize**

Description: **VE_GetSystemAttribute** allows configuration attributes of the VeriCall Edge system to be retrieved. This function may be used any time after the initialization of the VeriCall Edge system with the **VE_Initialize** function call.

The *ve_application_id* argument must be that of a previously registered application. The *ve_attribute_type*, used to identify the field to be set, must be one of the enumerated types defined in **t_ve_attribute_type_e**, as described in Appendix B.71 on Page 504. Associated VeriCall Edge structures to get these attributes are stated in Table C.1 on Page 659.

The *ve_attribute_index* argument is used to define an instance entry and should be unique for each instance. Associated attribute index values are stated in Table C.2 on Page 662.

Continued >>

The memory allocated in the pointer argument *ve_user_data*, must be of size *size_of_data* and be of equal size to the *ve_attribute_type* under consideration, otherwise the function will return the enumerated error code VE_RETURN_CODE_USER_ALLOC_TOO_SMALL.

A macro (VE_SINGLETON_DATA_ENTITY) is provided to define the *ve_attribute_index* argument for all singleton data entities. The provided macro is defined in Appendix D.3 on Page 668.

Example: This example shows how to get a system attribute from the framework attribute store:

```
#include "ve_eapi.h"

/* Get the system attribute for the G711 u-law configuration */
{
   t_ve_return_code_e ret_code;
   t_ve_media_definition_s media_definition;

   ret_code = VE_GetSystemAttribute(application_id
              VE_ATTRIBUTE_TYPE_MEDIA_PROFILE,
              G711U_MEDIA_ID,
              &media_definition,
              sizeof(t_ve_media_definition_s));

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_UpdateSystemAttribute

Syntax:

```
t_ve_return_code_e  VE_UpdateSystemAttribute(
                        UWord32 ve_application_id,
                        t_ve_attribute_type_e ve_attribute_type,
                        void *ve_user_data,
                        UWord32 size_of_data )
```

Arguments:

ve_application_id	The registered application ID.
ve_attribute_type	An enumerated value, which is used for classification of the data entity.
ve_user_data	Pointer to the data structure to be stored.
size_of_data	Size of the passed data structure.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_SetSystemAttribute**, **VE_GetSystemAttribute**, **VE_Initialize**

Description: **VE_UpdateSystemAttribute** allows configuration attributes of the VeriCall Edge system to be updated. This function may be used any time after the initialization of the VeriCall Edge system with the **VE_Initialize** function call. This function allows the VeriCall Edge Network module attributes to be updated post-commit (**VE_Commit**).

In the VeriCall Edge V2.2 release this allows the Type of Service (TOS) settings to be updated by the Network module. These alteration to the TOS settings will be activated across all the VeriCall Edge modules.

The *ve_application_id* argument must be that of a previously registered application. The *ve_attribute_type*, used to identify the field to be set, must be one of the enumerated types defined in **t_ve_attribute_type_e**, as described in Appendix B.71 on Page 504. Associated VeriCall Edge structures to get these attributes are stated in Table C.1 on Page 659.

The memory allocated in the pointer argument *ve_user_data*, must be of size *ve_size_of_data* and be of equal size to the *ve_attribute_type* under consideration, otherwise the function will return the enumerated error code VE_RETURN_CODE_USER_ALLOC_TOO_SMALL.

Continued >>

Example: This example shows how to update a system attribute:

```
#include "ve_eapi.h"

/* Update the system attribute for the SIP stack IP settings */
{
   t_ve_return_code_e ret_code;
   t_ve_ip_settings_default_configuration_s ip_settings;

   ip_settings.ve_tos = 0;

   ret_code = VE_UpdateSystemAttribute( application_id
                VE_ATTRIBUTE_TYPE_DEFAULT_SIP_IP_SETTINGS,
                &ip_settings,
                sizeof(
t_ve_ip_settings_default_configuration_s));

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_UpdateChannelAttribute

Syntax:

```
t_ve_return_code_e  VE_UpdateChannelAttribute(
                        UWord32 ve_application_id,
                        t_ve_attribute_type_e ve_attribute_type,
                        UWord32 channel,
                        void *ve_user_data,
                        UWord32 size_of_data )
```

Arguments:

ve_application_id	The registered application ID.
ve_attribute_type	An enumerated value, which is used for classification of the data entity.
channel	The channel identifier for which this action should be performed on.
ve_user_data	Pointer to the data structure to be stored.
size_of_data	Size of the passed data structure.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_SetSystemAttribute**, **VE_GetSystemAttribute**, **VE_Initialize**

Description: **VE_UpdateChannelAttribute** allows the channel configuration attributes to be updated. This function may be used any time after the initialization of the VeriCall Edge system with the **VE_Initialize** function call. This function allows the VeriCall Edge channel attributes to be updated post-commit (**VE_Commit**).

The *ve_application_id* argument must be that of a previously registered application. The *ve_attribute_type*, used to identify the field to be set, must be one of the enumerated types defined in **t_ve_attribute_type_e**, as described in Appendix B.71 on Page 504. Associated VeriCall Edge structures to get these attributes are stated in Table C.1 on Page 659.

The *channel* argument is used to identify the channel which should be updated. The memory allocated in the pointer argument *ve_user_data*, must be of size *size_of_data* and be of equal size to the *ve_attribute_type* under consideration, otherwise the function will return the enumerated error code VE_RETURN_CODE_USER_ALLOC_TOO_SMALL.

Continued >>

Example: This example shows how to update a system attribute:

```
#include "ve_eapi.h"

/* Update the AES configuration system attribute for a channel  */
{
   t_ve_return_code_e ret_code;
   t_ve_acoustic_ecan_configuration_s ecan_config;
   t_ve_acoustic_ecan_aes_configuration_s *aes_config;

   ecan_config.ve_acoustic_ecan_id = AES_ACOUSTIC_ECAN_ID;
   aes_config = &ecan_config.ve_acoustic_ecan_config.aes;

   aes_config->ecan_enable  = VE_DEFAULT_CONFIGURATION_TRUE;
   aes_config->prefilter_enable = VE_DEFAULT_CONFIGURATION_TRUE;
   aes_config->mode  = VE_ACOUSTIC_ECAN_AES_MODE_NORMAL;
   aes_config->bulk_delay  = 50;
   aes_config->post_gain  = 256;
   aes_config->test_mode  = 0;
   aes_config->write_audio  = VE_DEFAULT_CONFIGURATION_FALSE;

   ret_code = VE_UpdateChannelAttribute( application_id
            VE_ATTRIBUTE_TYPE_DEFAULT_ACOUSTIC_ECAN,
            channel_number,
            &ecan_config,
            sizeof( t_ve_acoustic_ecan_configuration_s));

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_RemoveSystemAttribute

Syntax:

```
t_ve_return_code_e  VE_RemoveSystemAttribute(
                        UWord32 ve_application_id,
                        t_ve_attribute_type_e ve_attribute_type,
                        UWord32 ve_attribute_index )
```

Arguments:

ve_application_id	The registered application ID.
ve_attribute_type	An enumerated value, which is used for classification of the data entity.
ve_attribute_index	A unique key value that is assigned for each instance of a data entity.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_SetSystemAttribute**, **VE_GetSystemAttribute**

Description: **VE_RemoveSystemAttribute** allows configuration attributes of the VeriCall Edge system to be removed. This function can only be used after the initialization of the VeriCall Edge system with the **VE_Initialize** function call and before the VeriCall Edge system has been committed with the **VE_Commit** function call.

Attributes removed from the VeriCall Edge system can be restored by using the VE_SetSystemAttribute function call, but only before the system is committed with the VE_Commit function call.

The *ve_application_id* argument must be that of a previously registered application. The *ve_attribute_type*, used to identify the attribute field to be removed, must be one of the enumerated types defined in **t_ve_attribute_type_e**, as described in Appendix B.71 on Page 504. Associated VeriCall Edge structures for these attributes are stated in Table C.1 on Page 659.

Continued >>

The *ve_attribute_index* argument is used to define an instance entry and should be unique for each instance. Associated attribute index values are stated in Table C.2 on Page 662.

A macro (`VE_SINGLETON_DATA_ENTITY`) is provided to define the *`ve_attribute_index`* argument for all singleton data entities. The provided macro is defined in Appendix D.3 on Page 668.

Example: This example shows how to remove a system attribute:

```
#include "ve_eapi.h"

/* Remove the system attribute for the H263 CIF media profile */
{
   t_ve_return_code_e ret_code;

   ret_code = VE_RemoveSystemAttribute( application_id
                VE_ATTRIBUTE_TYPE_MEDIA_PROFILE,
                H263_CIF_MEDIA_ID);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_Commit

Syntax:

```
t_ve_return_code_e  VE_Commit(
                        UWord32 ve_application_id,
                        UWord32 ve_max_context_size_in_bytes,
                        UWord32 *ve_max_memory_used_in_bytes )
```

Arguments:

ve_application_id	The registered primary application ID.
ve_max_context_size_in_bytes	The maximum size of context memory that the VeriCall Edge system is to use.
ve_max_memory_used_in_bytes	Pointer to storage of the actual memory context requirements of the VeriCall Edge system.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_Initialize**, **VE_SetSystemAttribute**, **VE_RemoveSystemAttribute**, **VE_GetSystemAttribute**, **VE_MemoryResourceQuery**

Description: **VE_Commit** applies the configuration entered during the initialization phase. After use of this function has been made, the VeriCall Edge system attributes cannot be altered through the use of **VE_SetSystemAttribute**, they can however still be read by the use of **VE_GetSystemAttribute**.

The *ve_max_context_size_in_bytes* argument is used to set the upper limit of the memory context size to be used by the VeriCall Edge system. A specification of zero will be classified as unrestricted, and thus the VeriCall Edge system will attempt to allocate all of the memory that it requires. If the memory specification is greater than zero, but smaller than the required amount, an error code will be returned to the caller. The function will return the actual amount used within the *ve_max_memory_used_in_bytes* argument. The current memory requirements of the VeriCall Edge system can be obtained through a call to **VE_MemoryResourceQuery**.

Continued >>

When the **VE_Commit** function is called, a calculation of the memory required by Edge will be calculated and returned to the caller via the parameter *ve_max_memory_used_in_bytes* argument.

If any parts of the commit or channel setup fail, the commit will be backed out, i.e. the VeriCall Edge system will not have been committed and an appropriate message will be added to the logging system.

Example: This example shows how to commit a VeriCall Edge system:

```
#include "ve_eapi.h"

/* Commit VeriCall Edge */
{
   t_ve_return_code_e ret_code;
   UWord32 bytes_required;
   UWord32 max_bytes_required = 0;

   /* Query VeriCall Edge for the amount of memory required */
   ret_code = VE_MemoryResourceQuery(&bytes_required);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Commit the VeriCall Edge system */
   ret_code = VE_Commit( applicationId,
       bytes_required,
       &max_context_size_in_bytes);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_RunVeriCallEdge

Syntax:

```
t_ve_return_code_e  VE_RunVeriCallEdge(
                                UWord32 ve_application_id,
                                t_ve_system_s *system_attrib )
```

Arguments:

ve_application_id	The registered application ID.
system_attrib	Pointer to a structure containing the number of queued messages available to this registered application.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_RunVeriCallEdgeWithConfig**

Description: **VE_RunVeriCallEdge** is called by the user application to execute all the media processing functions of the VeriCall Edge system. Because the VeriCall Edge system reads and writes to the user media interface it is necessary to call this function sufficiently often to prevent buffer overflows from the media device or buffer underflows to the media device. The first call to this function will complete the commit process.

The *ve_application_id* argument must be that of the application that initialized the system. The number of queued messages associated with this registered application is updated and placed in the structure pointed to by *system_attrib*. This structure, of type **t_ve_system_s**, is described in Appendix B.1 on Page 253.

A call to VE_RunVeriCallEdge will be rejected if VE_Commit is invalid or has not been performed.

Example: This example shows how to call **VE_RunVeriCallEdge**:

```
#include "ve_eapi.h"

/* Every "tick", this function should be called. */
/* The "tick" should occur on a regular and frequent basis */
{
   /* Variable declarations */
   t_ve_return_code_e ret_code;
   t_ve_system_s system_attrib;
```

Continued >>

```
    ret_code = VE_RunVeriCallEdge( application_id, &system_attrib );

    if(ret_code != VE_RETURN_CODE_SUCCESS)
    {
       /* Handle error */
    }
    else
    {
       /* Otherwise return the number of messages */
       if( num_messages != NULL )
       {
    num_messages = system_attrib.message_in_queue;
       }
       else
       {
    /* Handle no messages in queue */
       }
    }

    return ret_code;
}
```

VE_RunVeriCallEdgeWithConfig

Syntax:

```
t_ve_return_code_e  VE_RunVeriCallEdgeWithConfig(
                      UWord32 ve_application_id,
                      t_ve_run_vericall_edge_config_s *config,
                      t_ve_system_s *system_attrib )
```

Arguments:

ve_application_id	The registered application ID.
config	Pointer to a structure containing the configuration parameters with which VeriCall Edge should be run.
system_attrib	Pointer to a structure containing the number of queued messages available to this registered application.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_Initialize**, **VE_Commit**, **VE_Shutdown**

Description: **VE_RunVeriCallEdgeWithConfig** is similar to the **VE_RunVeriCallEdge** function and can be used instead of or intermittently with. This function allows the VeriCall Edge resource usage to be controlled.

This function can be used to place specified VeriCall Edge classes, where a class is defined as being an audio or video module, into what is termed low-power mode. Low power mode releases specific sections of the class from VeriCall Edge control. Use of this function can also be used to re-enable (awake) these classes from their low power mode.

This function can be used intermittently with **VE_RunVeriCallEdge**, alternatively it can be used in place of **VE_RunVeriCallEdge**. When used in place of **VE_RunVeriCallEdge** the configuration settings can be set not to enable low power mode.

The *ve_application_id* argument must be that of the application that initialized the system. The configuration structure to be used with this API function is pointed to by *config*. The **t_ve_run_vericall_edge_config_s** structure, described in Appendix B.6 on Page 258, is used to define the VeriCall Edge classes to be run.

Continued >>

The number of queued messages associated with this registered application is updated and placed in the structure pointed to by *system_attrib*. This structure, of type **t_ve_system_s**, is described in Appendix B.1 on Page 253.

A call to VE_RunVeriCallEdgeWithConfig will be rejected if VE_Commit is invalid or has not been performed.

Example: This example shows how to place the audio and video classes into an *idle* state:

```
#include "ve_eapi.h"

/* Place the audio and video modules into the inactive state */
/* If there is a call running this request will be rejected */
{
   t_ve_return_code_e ret_code;
   t_ve_system_s system_attrib;
   t_ve_run_vericall_edge_config_s run_ve_config;

   run_ve_config.class_select = VE_MODULE_CLASS_AUDIO_VIDEO;
   run_ve_config.module_state_class.audio_video_module_state.audio
=
      VE_MODULE_STATE_IDLE;

run_ve_config.module_state_class.audio_video_module_state.video_dec
oder =
      VE_MODULE_STATE_IDLE;

run_ve_config.module_state_class.audio_video_module_state.video_pre
_processor =
      VE_MODULE_STATE_IDLE;

run_ve_config.module_state_class.audio_video_module_state.video_enc
oder =
      VE_MODULE_STATE_IDLE;

run_ve_config.module_state_class.audio_video_module_state.video_pos
t_processor =
      VE_MODULE_STATE_IDLE;

   ret_code = VE_RunVeriCallEdgeWithConfig( applicationId,
               &run_ve_config,
               &system_attrib);
```

Continued >>

```
   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   else
   {
      /* Otherwise return the number of messages */
      if( num_messages != NULL )
      {
   num_messages = system_attrib.message_in_queue;
      }
      else
      {
   /* Handle no messages in queue */
      }
   }

   return ret_code;
}
```

Example: This example shows how to place the audio and video classes into an *active* state:

```
#include "ve_eapi.h"

/* Place the audio and video modules into the active state */
{
   t_ve_return_code_e ret_code;
   t_ve_system_s system_attrib;
   t_ve_run_vericall_edge_config_s run_ve_config;

   run_ve_config.class_select = VE_MODULE_CLASS_AUDIO_VIDEO;
   run_ve_config.module_state_class.audio_video_module_state.audio
=
      VE_MODULE_STATE_ACTIVE;

run_ve_config.module_state_class.audio_video_module_state.video_pre
_processor =
      VE_MODULE_STATE_ACTIVE;

run_ve_config.module_state_class.audio_video_module_state.video_enc
oder =
      VE_MODULE_STATE_ACTIVE;

run_ve_config.module_state_class.audio_video_module_state.video_pos
t_processor =
      VE_MODULE_STATE_ACTIVE;

run_ve_config.module_state_class.audio_video_module_state.video_dec
oder =
      VE_MODULE_STATE_ACTIVE;
```

Continued >>

```
    ret_code = VE_RunVeriCallEdgeWithConfig(applicationId,
                &run_ve_config,
                &system_attrib);

    if(ret_code != VE_RETURN_CODE_SUCCESS)
    {
       /* Handle error */
    }
    else
    {
       /* Otherwise return the number of messages */
       if( num_messages != NULL )
       {
    num_messages = system_attrib.message_in_queue;
       }
       else
       {
    /* Handle no messages in queue */
       }
    }

    return ret_code;
}
```

VE_Shutdown

Syntax: `t_ve_return_code_e VE_Shutdown( UWord32 ve_application_id )`

Arguments: *ve_application_id* The registered primary application ID.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_Initialize**

Description: **VE_Shutdown** will make a request to the Media Engine and call control stack to shutdown and release allocated resources. When the VeriCall Edge system has completed its shutdown process, the user application will be notified by the shutdown complete message.

The *ve_application_id* argument must be that of the primary application that initialized the system.

This function must be used to stop VeriCall Edge as it allows the VeriCall Edge system to close all the resources it has used cleanly. Using methods such as CTRL+C to stop the VeriCall Edge application will result in such things as the audio driver not being closed correctly.

The user application should wait for the shutdown complete message before allowing the VeriCall Edge user application to exit. The shutdown complete message, VE_USER_MSG_SHUTDOWN_COMPLETE is described in Appendix B.33 on Page 356.

If there are any active calls on the VeriCall Edge system when this function is used, the active calls will be terminated.

Continued >>

Example: This example shows use of the **VE_Shutdown** function:

```
#include "ve_eapi.h"

/* ONLY shutdown the VeriCall Edge system using this function */
{
   t_ve_return_code_e ret_code;

   ret_code = VE_Shutdown( ve_application_id );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Wait for VE_USER_MSG_SHUTDOWN_COMPLETE message here */
   /* ONLY when this message is received, clear down application
resources
      /* and exit normally */
}
```

VE_GetNextEvent

Syntax:

```
t_ve_return_code_e  VE_GetNextEvent(
                                    UWord32 ve_application_id,
                                    t_ve_user_msg_s *ve_event )
```

Arguments:

ve_application_id	The registered application ID.
ve_event	Pointer to the structure used to store the event information.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_GetNextEvent** is used by the application to get an event that is queued. The *ve_application_id* argument must be valid as this is used to determine if the event message is for the application that called the function.

The event information is retrieved and placed in the structure pointed to by *ve_event*. This structure of type **t_ve_user_msg_s**, is described in Appendix B.33 on Page 356. If no messages are available for the specified application, the return code VE_RETURN_CODE_NO_MESSAGES will be returned to the caller.

Example: This example shows how to get an event from the VeriCall Edge system:

```
#include "ve_eapi.h"

/* Register the application to get an application ID */
{
   t_ve_return_code_e ret_code;
   t_ve_user_msg_s    event

      ret_code = GetNextEvent(application_id, &event);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_GetVersion

Syntax: **`t_ve_return_code_e VE_GetVersion( t_ve_version_s *`***`ve_version`***` )`**

Arguments: *ve_version* Pointer to a structure used to store the version information.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_GetVersion** is used to get the version number of the VeriCall Edge system software. The version information is retrieved and placed in the structure pointed to by *ve_version*. This structure of type **t_ve_version_s**, is described in Appendix B.3 on Page 255.

Note that a common software version number is shared across all the VeriCall Edge software modules. The actual returned version number is configured at build time.

Example: This example shows one method of using this function:

```
#include "ve_eapi.h"
#include <stdio.h>

{
   t_ve_version_s  version;
   t_ve_return_code_e ret_code;

   ret_code = VE_GetVersion( &version );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   printf("VeriCall Edge version %lu.%lu.%lu (%s)\n",
                  version.major, version.minor,
                  version.patch, version.qualifier );
}
```

VE_GetStatistics

Syntax:

```
t_ve_return_code_e  VE_GetStatistics(
                      UWord32 ve_application_id,
                      t_ve_stats_type_e ve_stats_type,
                      UWord32 ve_stats_instance,
                      t_ve_statistic_u *ve_return_stats_data )
```

Arguments:

ve_application_id	The registered application ID.
ve_stats_type	An enumerated value indicating the statistics type for this function call.
ve_stats_instance	Used to determine the instance of statistics to obtain.
ve_return_stats_data	A union of statistic elements used for the storage of the statistics information.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_ClearStatistics**.

Description: **VE_GetStatistics** is used to request statistics from the VeriCall Edge system. All the VeriCall Edge statistics are updated on a periodic basis. The *ve_application_id* argument must be a valid registered application identifier.

The argument *ve_stats_type* must be an enumerated type, defined in **t_ve_stats_type_e** and used to indicate which statistics type is being requested and is described in Appendix B.179 on Page 600. If statistics are gathered on a channel basis, *ve_stats_instance* should be the appropriate channel identification key.

If the statistics type is a system wide statistic, then *ve_stats_instance* should be set to be VE_SINGLETON_DATA_ENTITY. The combination of *ve_stats_type* and *ve_stats_instance* form the unique key to reference the appropriate statistics from the VeriCall Edge system.

Continued >>

The requested statistics information is retrieved and placed into the allocated area of memory pointed to by *ve_return_stats_data*. This is a union of structures, **t_ve_statistic_u** as described in Appendix B.38 on Page 431, with each statistic structure being called based on the value used in the *ve_stats_type* argument.

Note that the periodic update of statistics is determined by the periodic calling of VE_RunVeriCallEdge/VE_RunVeriCallEdgeWithConfig, every call to either of these functions results in a statistics update.

Example: This example shows hw to obtain the system level statistics:

```
#include "ve_eapi.h"
#include <stdio.h>

{
   t_ve_statistic_u stat_info;
   t_ve_return_code_e rc;

   /* Request Statistic information from VeriCall Edge -
      Get VE_STATISTICS_TYPE_SYSTEM_CALLS obtained from ve_eapi.h
      These are system wide statistics so there can only be
      one instance. We should use VE_SINGLETON_DATA_ENTITY
      when there is only one instance */

   if(VE_GetStatistics(application_id,
         VE_STATISTICS_TYPE_SYSTEM_CALLS,
         VE_SINGLETON_DATA_ENTITY,
         (t_ve_statistic_u*) &stat_info) != VE_RETURN_CODE_SUCCESS)
   {
      /* Statistic has not been retrieved  - handle error here */
   }
   else
   {
      /* Display retrieved statistics */
      printf("Display Statistics: Completed calls = %lu\n",
                           stat_info.system_calls.completed_calls);
      printf("Display Statistics: Failed calls = %lu\n",
                           stat_info.system_calls.failed_calls);
      printf("Display Statistics: Busy calls = %lu\n",
                           stat_info.system_calls.busy_calls);
   }
}
```

VE_ClearStatistics

Syntax:

```
t_ve_return_code_e  VE_ClearStatistics(
                                UWord32 ve_application_id,
                                t_ve_stats_type_e ve_stats_type,
                                UWord32 ve_stats_instance )
```

Arguments:

ve_application_id	The registered application ID.
ve_stats_type	An enumerated value indicating the statistics type for this function call.
ve_stats_instance	Used to determine the instance of statistics to clear.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_GetStatistics**.

Description: **VE_ClearStatistics** is used to reset or clear the per-channel and system statistics in the VeriCall Edge system. All VeriCall Edge system statistics are updated on a periodic basis.

The *ve_application_id* argument must be a valid registered application identifier. The argument *ve_stats_type* must be an enumerated type, defined in **t_ve_stats_type_e**, and used to indicate which statistics type is being requested and is described in Appendix B.179 on Page 600. If per-channel statistics are to be cleared, *ve_stats_instance* should be the appropriate channel identification key.

If the statistics type is a system wide statistic, then *ve_stats_instance* should be set to be VE_SINGLETON_DATA_ENTITY. The combination of *ve_stats_type* and *ve_stats_instance* form the unique key to reference the appropriate statistics from the VeriCall Edge system.

Note that the RTCP based statistics, as described in t_ve_stats_type_e in Appendix B.179 on Page 600, cannot be reset, as these fields are under the control of the RTP session.

Continued >>

Example: This example shows how to clear the system statistics:

```
#include "ve_eapi.h"

{
   t_ve_return_code_e rc;

   /* Clear Statistic information in VeriCall Edge -
      Clear VE_STATISTICS_TYPE_SYSTEM_CALLS obtained from ve_eapi.h
      These are system wide statistics so there can only be
      one instance. We should use VE_SINGLETON_DATA_ENTITY
      when there is only one instance */

   rc = VE_ClearStatistics( application_id,
                            VE_STATISTICS_TYPE_SYSTEM_CALLS,
                            VE_SINGLETON_DATA_ENTITY);

   if( rc != VE_RETURN_CODE_SUCCESS)
   {
      /* Statistic have not been cleared  - Handle error here */
   }
}
```

Chapter 4

VeriCall Edge™ User API

This chapter describes the VeriCall Edge User API functions. The VeriCall Edge User API is used for user administration.

4.1 VeriCall Edge User API Function Descriptions

For each function detailed in this section, the following are described (as appropriate):

- **Syntax** Command syntax.
- **Arguments** Description of the arguments.
- **Returns** Return codes from the function.
- **See Also** Other functions that perform related tasks.
- **Description** What the function can be used for.
- **Example** Short code sections showing how to use the function.

4.2 VeriCall Edge User API Functions

The VeriCall Edge User API functions are described on the following pages:

Function	Page
VE_AddUser	41
VE_UpdateUser	43
VE_RemoveUser	45
VE_UpdateUserAttribute	47

VE_AddUser

Syntax:

```
t_ve_return_code_e  VE_AddUser(
                    UWord32 ve_application_id,
                    t_ve_user_channel_details_s *ve_user_details,
                    t_ve_user_handle_s *ve_user_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_details	Pointer to structure containing channel information of the user to be added.
ve_user_handle	Pointer to the structure to store the handle of the added user.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_UpdateUser**, **VE_RemoveUser**, **VE_SIPRegisterUser**, **VE_SIPDeregisterUser**, **VE_H323RegisterUser**, **VE_H323DeregisterUser**

Description: **VE_AddUser** is used to define a user with the VeriCall Edge system for the first time. The *ve_application_id* argument must be a valid registered application identifier. The channel information of the user is contained in the structure pointed to by *ve_user_details*. This structure of type **t_ve_user_channel_details_s**, is described in Appendix B.26 on Page 300. The handle allocated to the newly added user is stored in the structure pointed to by *ve_user_handle*. This structure of type **t_ve_user_handle_s** is described in Appendix B.25 on Page 299. This user handle must be used as the call specifier for all subsequent call actions.

The use of this function defines a user with the VeriCall Edge system call control stack(s). To register a user with an external SIP registration server, the VE_SIPRegisterUser function should be used. Alternatively, to register a user with a H.323 gatekeeper, the VE_H323RegisterUser function call should be used.

Continued >>

Example: This example shows how to add a user o the VeriCall Edge system:

```
#include "ve_eapi.c"

/* Add a user to a single audio only channel */
{
   t_ve_user_channel_details_s user_details;
   t_ve_return_code_e result;

   user_details.number_of_channels = 1; /* Assuming channel 1 */
   user_details.channel_association[0].channel_selection.
                                                channel_number = 1;
   /* No 3-way calling involved with this channel */
   user_details.channel_association[0].channel_selection.
          sub_channel_select= VE_USER_SUB_CHANNEL_SELECT_PRIMARY;

   /* Set up the preferences for this users audio channel.
      Preferences are used by the negotiation stack to identify
      preference of choice. The first in the list will be
      considered first choice, second in the list second choice and
      so on */
   /* We are only considering 3 audio algorithms in this ex maple.
      Although we only have 3 voice algorithms, we also need to
      include CN. It is a separate entity as we will need to
      negotiate this as a separate line item in the call
      negotiation */
   user_details.channel_association[0].num_algs_in_pref_list = 4;
   user_details.channel_association[0].alg_preference[0]=
                                                G711U_MEDIA_ID;
   user_details.channel_association[0].alg_preference[1]=
                                                G711A_MEDIA_ID;
   user_details.channel_association[0].alg_preference[2]=
                                                G729_AB_MEDIA_ID;
   user_details.channel_association[0].alg_preference[3]=
                                                CN_MEDIA_ID;

   /* Add the user to the VeriCall Edge System */
   result = VE_AddUser( applicationId,
            &user_details,
            &ve_user_handle);

   if( result != VE_RETURN_CODE_SUCCESS )
   {
      /* Handle error here */
   }

   /* Before this user can make calls, the user must have a
      SIP or H.323 identity, via VE_SIPAddUsersRegistrationData or
      VE_H323AddUsersRegistrationData */
}
```

VE_UpdateUser

Syntax:

```
t_ve_return_code_e  VE_UpdateUser(
                    UWord32 ve_application_id,
                    t_ve_user_channel_details_s *ve_user_details,
                    t_ve_user_handle_s *ve_user_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_details	Pointer to structure containing channel information of the user to be updated.
ve_user_handle	Pointer to structure containing the handle of the user to be updated.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_AddUser**, **VE_RemoveUser**, **VE_SIPRegisterUser**, **VE_SIPDeregisterUser**, **VE_H323DeregisterUser**, **VE_H323RegisterUser**

Description: **VE_UpdateUser** is used to update information for a user previously defined with **VE_AddUser**. The *ve_application_id* argument must be a valid registered application identifier. The updated user channel information is contained in the structure pointed to by *ve_user_details*. This structure of type **t_ve_user_channel_details_s** is described in Appendix B.26 on Page 300. The handle associated with the user to be updated is contained in the structure pointed to by *ve_user_handle*. This structure of type **t_ve_user_handle_s** is described in Appendix B.25 on Page 299.

Example: This example updates a users preference list:

```
#include "ve_eapi.c"

/* Update a user's algorithm preference list for a single audio
   only channel */
/* Normally user application would have a local copy of the users
   details, or we could get the information from the framework data
   store (attribute) */
{
   t_ve_user_channel_details_s user_details;
   t_ve_return_code_e result;
```

Continued >>

```
    /* Assume still channel 1 */
    user_details.number_of_channels = 1;
    user_details.channel_association[0].channel_selection.
                                                channel_number = 1;
    /* Still no 3-way calling involved with this channel */
    user_details.channel_association[0].channel_selection.
          sub_channel_select= VE_USER_SUB_CHANNEL_SELECT_PRIMARY;

    /* Add G.723_1 to user's preferences and make it their preferred
       choice */
    user_details.channel_association[0].num_algs_in_pref_list = 4;
    user_details.channel_association[0].alg_preference[0]=
                                                G723_1_MEDIA_ID;
    user_details.channel_association[0].alg_preference[1]=
                                                G711U_MEDIA_ID;
    user_details.channel_association[0].alg_preference[2]=
                                                G711A_MEDIA_ID;
    user_details.channel_association[0].alg_preference[3]=
                                                G729_AB_MEDIA_ID;
    user_details.channel_association[0].alg_preference[4]=
                                                CN_MEDIA_ID;

    /* Update the user's details to the VeriCall Edge System */
    result = VE_UpdateUser( applicationId,
          &user_details,
          &ve_user_handle);

    if( result != VE_RETURN_CODE_SUCCESS )
    {
       /* Handle error here */
    }

    /* Before this user can make calls using the updated preference
       list, their SIP or H.323 identity has to be updated via
       VE_SIPUpdateUsersRegistrationData or
       VE_H323UpdateUsersRegistrationData */
}
```

VE_RemoveUser

Syntax:

```
t_ve_return_code_e  VE_RemoveUser(
                        UWord32 ve_application_id,
                        t_ve_user_handle_s *ve_user_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to structure containing the handle of the user to be removed.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_AddUser**, **VE_UpdateUser**, **VE_SIPRegisterUser**, **VE_SIPDeregisterUser**, **VE_H323DeregisterUser**, **VE_H323RegisterUser**

Description: **VE_RemoveUser** is used to remove a user from the VeriCall Edge system. The *ve_application_id* argument must be a valid registered application identifier. The handle associated with the user to be removed is contained in the structure pointed to by *ve_user_handle*. This structure of type **t_ve_user_handle_s** is described in Appendix B.25 on Page 299.

The use of this function call will immediately remove a user and disassociate that user from any channels allocated to that user.

The use of this function only removes a user from the VeriCall Edge system. To deregister the user from a SIP registration server or H.323 gatekeeper, the VE_SIPDeregisterUser and VE_H323DeregisterUser functions must be used before VE_RemoveUser.

Continued >>

Example: This example shows how to de-register a user from the VeriCall Edge system:

```
#include "ve_eapi.c"

/* Remove a user from the VeriCall Edge system */
{
   t_ve_return_code_e result;

   /* We should first de-register the user from the remote
      SIP/H.323 server */
   /* We will assume the user is a SIP user */
   result = VE_SIPDeregisterUser( applicationId,
                                  VE_SIP_DEREGISTER_LOCAL_CONTACTS,
                                  &ve_user_handle );

   if( result != VE_RETURN_CODE_SUCCESS )
   {
      /* Handle error here */
   }

   /* Next we should remove the user's registration data */
   result = VE_SIPRemoveUsersRegistrationData( applicationId,
                                               &ve_user_handle );

   if( result != VE_RETURN_CODE_SUCCESS )
   {
      /* Handle error here */
   }

   /* Finally, remove the user from VeriCall Edge */
   result = VE_RemoveUser( applicationId, &ve_user_handle );

   if( result != VE_RETURN_CODE_SUCCESS )
   {
      /* Handle error here */
   }
}
```

VE_UpdateUserAttribute

Syntax:

```
t_ve_return_code_e  VE_UpdateUserAttribute(
                     UWord32 ve_application_id,
                     t_ve_attribute_type_e ve_attribute_type,
                     t_ve_user_handle_s *ve_user_handle,
                     void *ve_user_data,
                     UWord32 size )
```

Arguments:

ve_application_id	The registered application ID.
ve_attribute_type	An enumerated value, which is used for classification of the data entity.
ve_user_handle	Pointer to structure containing the handle of the user to be removed.
ve_user_data	Pointer to the data structure to be stored.
size	Size of the passed data structure.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_AddUser**, **VE_UpdateUser**, **VE_SIPRegisterUser**, **VE_SIPDeregisterUser**, **VE_H323DeregisterUser**, **VE_H323RegisterUser**

Description: **VE_UpdateUserAttribute** can be used to update the user attributes stored within the VeriCall Edge system. This function may be used any time after the initialization of the VeriCall Edge system with the **VE_Initialize** function call.

The *ve_application_id* argument must be that of a previously registered application. The *ve_attribute_type*, used to identify the field to be set, must be one of the enumerated types defined in **t_ve_attribute_type_e**, as described in Appendix B.71 on Page 504. Associated VeriCall Edge structures to get these attributes are stated in Table C.1 on Page 659.

The handle associated with the user whose attributes are to be updated is contained in the structure pointed to by *ve_user_handle*. This structure of type **t_ve_user_handle_s** is described in Appendix B.25 on Page 299.

The memory allocated in the pointer argument *ve_user_data*, must be of size *size* and be of equal size to the attribute type under consideration, otherwise the function will return the enumerated error code VE_RETURN_CODE_USER_ALLOC_TOO_SMALL.

Continued >>

Example: This simple example shows how to use the **VE_UpdateUserAttribute** function:

```
#include "ve_eapi.h"

/* Update the system IP settings */
{
   t_ve_return_code_e ret_code;
   t_ve_ip_settings_default_configuration_s ip_settings;

   ip_settings.ve_tos = 0;

   /* Use VE_UpdateSystemAttribute as we are now post-commit so
      cannot use VE_SetSystemAttribute */
   ret_code = VE_UpdateUserAttribute( application_id
               VE_ATTRIBUTE_TYPE_DEFAULT_SYSTEM_IP_SETTINGS,
               &user_handle,
               &ip_settings,
               sizeof( t_ve_ip_settings_default_configuration_s));

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

Chapter 5

VeriCall Edge™ Media Control API

This chapter describes the VeriCall Edge Media Control API functions. The VeriCall Edge Media Control API is used for configuring, enabling and disabling the VeriCall Edge system Media Control functionality.

5.1 VeriCall Edge Media Control API Function Descriptions

For each function detailed in this section, the following are described (as appropriate):

- **Syntax** Command syntax.
- **Arguments** Description of the arguments.
- **Returns** Return codes from the function.
- **See Also** Other functions that perform related tasks.
- **Description** What the function can be used for.
- **Example** Short code sections showing how to use the function.

5.2 VeriCall Edge Media Control Functions

The VeriCall Edge Media Control functions are described on the following pages:

Function	Page
VE_StartStream	51
VE_StopStream	53
VE_UpdateStream	55
VE_GenerateNamedTone	58
VE_GenerateUserTone	62
VE_GenerateDigitString	66
VE_ControlAudio	69
VE_ControlVideo	71
VE_ControlNetwork	73
VE_NETLookUp	75

VE_StartStream

Syntax:

```
t_ve_return_code_e  VE_StartStream(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_stream_configuration_s *ve_stream_config )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user associated with the channel.
ve_stream_config	Pointer to the structure containing the call configuration parameters.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_StopStream**, **VE_GetNextEvent**, , **VE_H323StartCall**

Description: **VE_StartStream** is used to setup and start the media stream. The way that the structure pointed to by *ve_stream_config* is derived is dependent upon whether the calling application is using the VeriCall Edge internal call control stack or an external third-party call control stack.

If the function call is to be used with the VeriCall Edge software SIP or H.323 stack, the user application will invoke the **VE_SIPSessionStart** or **VE_H323StartCall** function, before using this function call, which will negotiate the channel configuration. This information will be sent to the user application via the structure associated with the SIP message, VE_USER_MSG_SIP_SESSION_STARTED, or the H.323 message, VE_USER_MSG_H323_CALL_STARTED, (obtained via **VE_GetNextEvent**) and will be encapsulated in a **t_ve_stream_configuration_s** structure as described in Appendix B.24 on Page 298. This structure can then be passed to **VE_StartStream** as the call configuration attribute.

If the caller of **VE_StartStream** is not using the VeriCall Edge software SIP or H.323 stack, it is the responsibility of the calling user application to complete the **t_ve_stream_configuration_s** structure.

Continued >>

The call configuration structure is pointed to by *ve_stream_config*. This structure of type **t_ve_stream_configuration_s**, is described in Appendix B.24 on Page 298. The *ve_user_handle* argument points to the **t_ve_user_handle_s** structure used to identify the user, from which the channel can be derived and is described in Appendix B.25 on Page 299. The *ve_application_id* argument must be a valid registered application identifier.

Example: This simple example shows the use of this function:

```
#include "ve_eapi.c"

/* Assume a VE_USER_MSG_SIP_SESSION_STARTED event has been
   received */
{
   t_ve_return_code_e result;
   t_ve_user_msg_s    ve_event;

   /* Switch off tone generation */
   result = VE_GenerateNamedTone( applicationId,
           &ve_event->message.sip_session_started.ve_user_handle,
           VE_COUNTRY_TYPE_UK,
           VE_TONE_NONE,
           VE_TONE_DIRECTION_LOCAL);

   if( result != VE_RETURN_CODE_SUCCESS )
   {
      /* Handle error here */
   }

   /* Start the data stream(s) we have negotiated in the media
      engine */
   /* The stream configuration structure is that which was
      negotiated */
   result = VE_StartStream( applicationId,
          &ve_event->message.sip_session_started.ve_user_handle,
          &ve_event->message.sip_session_started.ve_stream_config
);

   if( result != VE_RETURN_CODE_SUCCESS )
   {
      /* Handle error here */
   }
}
```

VE_StopStream

Syntax:

```
t_ve_return_code_e  VE_StopStream(
                        UWord32 ve_application_id,
                        t_ve_user_handle_s *ve_user_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user associated with the channel.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_SIPSessionStop**, **VE_StartStream**, **VE_H323StopCall**, **VE_UpdateStream**

Description: **VE_StopStream** is used to stop the identified media stream. The *ve_application_id* argument must be a valid registered application identifier. The user identification handle is contained in the structure pointed to by *ve_user_handle*, from which the channel number can be determined by the function. This structure of type **t_ve_user_handle_s** is described in Appendix B.25 on Page 299.

If the VeriCall Edge software SIP stack is being used, this function call needs to be made after a call to the **VE_SIPSessionStop** function. If the VeriCall Edge software H.323 call stack is being used, this function call needs to be made after a call to the **VE_H323StopCall** function.

Using this function will cause all the components of the stream to be stopped. This means that if there is a video and audio channel associated with the user, using this function will stop both channels. If only the audio channel as required, the video could only be stopped separately by using the **VE_UpdateStream** function.

Example: This simple example shows the use of this function:

```
#include "ve_eapi.c"

/* Assume a VE_USER_MSG_SIP_SESSION_STOPPED event (BYE) has been
   received */
{
   t_ve_return_code_e result;
   t_ve_user_msg_s    ve_event;
```

Continued >>

```
    /* Instruct VeriCall Edge to de-activate the RTP stream(s) */
    result = VE_StopStream( applicationId,
          &ve_event->message.sip_session_stopped.ve_user_handle );

    if( result != VE_RETURN_CODE_SUCCESS )
    {
       /* Handle error here */
    }
}
```

VE_UpdateStream

Syntax:

```
t_ve_return_code_e  VE_UpdateStream(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_stream_configuration_s *ve_stream_config )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user associated with the channel.
ve_stream_config	Pointer to the structure containing the updated calls configuration parameters.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_StartStream**, **VE_StopStream**, **VE_GetNextEvent**

Description: **VE_UpdateStream** is used to update the configuration of a media stream. The way that the updated structure pointed to by *ve_stream_config* is derived is dependent upon whether the calling application is using the VeriCall Edge software SIP/H.323 stack or an external third-party call control stack.

When using a third party call control stack, it is the responsibility of that stack to complete structure pointed to by *ve_stream_config*.

The call configuration structure is pointed to by *ve_stream_config*. This structure of type **t_ve_stream_configuration_s** is described in Appendix B.24 on Page 298. The *ve_user_handle* argument points to the **t_ve_user_handle_s** structure used to identify the user and is described in Appendix B.25 on Page 299. The *ve_application_id* argument must be a valid registered application identifier.

Continued >>

Example: This example shows how to use this function to update a stream:

```
#include "ve_eapi.c"

/* Assume a VE_USER_MSG_SIP_SESSION_STARTED event has been
   received */
{
   t_ve_return_code_e result;
   t_ve_user_msg_s  ve_event;

   /* Switch off tone generation */
   result = VE_GenerateNamedTone( applicationId,
           &ve_event->message.sip_session_started.ve_user_handle,
           VE_COUNTRY_TYPE_UK,
           VE_TONE_NONE,
           VE_TONE_DIRECTION_LOCAL);

   if( result != VE_RETURN_CODE_SUCCESS )
   {
      /* Handle error here */
   }

   /* Start the data stream(s) we have negotiated in the media
      engine */
   /* The stream configuration structure is that which was
      negotiated */
   result = VE_StartStream( applicationId,
          &ve_event->message.sip_session_started.ve_user_handle,
          &ve_event->message.sip_session_started.ve_stream_config
);

   if( result != VE_RETURN_CODE_SUCCESS )
   {
      /* Handle error here */
   }

   /* At some point later, at least after 'early media' */

   /* Remove the video channel = privacy */
   if(ve_event->message.sip_session_started.ve_stream_config.
                                          num_media_descriptors > 1)
      ve_event->message.sip_session_started.
                           ve_stream_config.media_descriptor[1].
                           rtpmapping_attrib.num_rtp_maps = 0;
```

Continued >>

```
    /* Assume the media engine is already active */
    /* Update the stream */
    result = VE_UpdateStream(applicationId,
          &ve_event->message.sip_session_started.ve_user_handle,
          &ve_event->message.sip_session_started.ve_stream_config);

    if( result != VE_RETURN_CODE_SUCCESS )
    {
       /* Handle error here */
    }
}
```

VE_GenerateNamedTone

Syntax:

```
t_ve_return_code_e  VE_GenerateNamedTone(
                      UWord32 ve_application_id,
                      t_ve_user_handle_s *ve_user_handle,
                      t_ve_country_e ve_country,
                      t_ve_tone_type_e ve_tone_type,
                      t_ve_tone_direction_e ve_tone_direction )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user associated to the channel.
ve_country	Defines the country that the tone should be generated for.
ve_tone_type	Defines the tone type to be generated.
ve_tone_direction	Defines the direction this tone should be played into the audio stream.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_GenerateDigitString**

Description: **VE_GenerateNamedTone** is used to generate a user specified (named) tone on the channel(s) associated to the user specified by *ve_user_handle*.

The *ve_application_id* argument must be a valid registered application identifier. The user identification handle is contained in the structure pointed to by *ve_user_handle*, from which the channel number can be determined by the function. This structure of type **t_ve_user_handle_s** is described in Appendix B.25 on Page 299.

Country specific versions of tones are set by *ve_country* that is an enumerated type, **t_ve_country_e,** as described in Appendix B.79 on Page 518. The tone to be generated is indicated by *ve_tone_type* and must be one of the enumerated types defined in **t_ve_tone_type_e** as described in Appendix B.78 on Page 517.

Continued >>

The direction that the tone is to be played into the audio stream is defined by the *ve_tone_direction* argument. This is an enumerated type, **t_ve_tone_direction_e** as described in Appendix B.79 on Page 518, and allows the generated tone to be played to the local or far end.

The transmission direction enumerated types can be OR'd together to allow bi-directional transmission of the digit string. Refer to the provided example code of this function to see an instance of this.

This function can only be used if the `named_tone_generator_enable` element of the t_ve_channel_s structure, described in Appendix B.21 on Page 289, has been enabled.

Example: This example shows different configurations of this function:

```
#include "ve_eapi.c"

{
   /* Switch on some state */
   switch(menu_state)
   {
   case MENU_TONE_DIAL_REMOTE:
      {
   /* Generate Dial Tone to the remote end */
   result = VE_GenerateNamedTone( applicationId,
                &ve_user_handle,
                VE_COUNTRY_TYPE_UK,
                VE_TONE_DIALTONE,
                VE_TONE_DIRECTION_REMOTE);

   if( result != VE_RETURN_CODE_SUCCESS )
   {
      /* Handle error here */
   }
      }
      break;

   case MENU_TONE_DIAL_LOCAL:
      {
   /* Generate Dial Tone to the local end */
   result = VE_GenerateNamedTone( applicationId,
                &ve_user_handle,
                VE_COUNTRY_TYPE_UK,
                VE_TONE_DIALTONE,
                VE_TONE_DIRECTION_LOCAL);
```

Continued >>

```
   if( result != VE_RETURN_CODE_SUCCESS )
   {
      /* Handle error here */
   }
      }
      break;

   case MENU_TONE_DIAL_BOTH:
      {
   /* Generate Dial Tone to local & remote ends */
   result = VE_GenerateNamedTone( applicationId,
                &ve_user_handle,
                VE_COUNTRY_TYPE_UK,
                VE_TONE_DIALTONE,

(VE_TONE_DIRECTION_LOCAL|VE_TONE_DIRECTION_REMOTE));

   if( result != VE_RETURN_CODE_SUCCESS )
   {
      /* Handle error here */
   }
      }
      break;

   case MENU_TONE_NONE_BOTH:
      {
   /* Stop any tones playing on local & remote ends */
   result = VE_GenerateNamedTone( applicationId,
                &ve_user_handle,
                VE_COUNTRY_TYPE_UK,
                VE_TONE_NONE,

(VE_TONE_DIRECTION_LOCAL|VE_TONE_DIRECTION_REMOTE));

   if( result != VE_RETURN_CODE_SUCCESS )
   {
      /* Handle error here */
   }
      }
      break;

   case MENU_TONE_NONE_LOCAL:
      {
   /* Stop any tones playing on local end */
   result = VE_GenerateNamedTone( applicationId,
                &ve_user_handle,
                VE_COUNTRY_TYPE_UK,
                VE_TONE_NONE,
                VE_TONE_DIRECTION_LOCAL);
```

Continued >>

```
        if( result != VE_RETURN_CODE_SUCCESS )
        {
           /* Handle error here */
        }
           }
           break;

           /* Other cases can be handled here */

        default:
           {
        /* Handle default case here */
           }
           break;
        }
    }
```

VE_GenerateUserTone

Syntax:

```
t_ve_return_code_e VE_GenerateUserTone(
                        UWord32 ve_application_id,
                        t_ve_user_handle_s *ve_user_handle,
                        t_ve_tone_definition_s *ve_tone,
                        t_ve_tone_direction_e ve_tone_direction )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user associated to the channel.
ve_tone	Structure that defines the tone to be generated.
ve_tone_direction	Defines the direction this tone should be played into the audio stream.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_GenerateDigitString**

Description: **VE_GenerateUserTone** is used to generate a user defined tone on the channel(s) associated to the user specified by *ve_user_handle*.

The user defined tone to be generated is defined by the structure pointed to by *ve_tone*. This structure, **t_ve_tone_definition_s** as described in Appendix B.59 on Page 488, allows the configuration and determination of the tone to be generated.

The *ve_application_id* argument must be a valid registered application identifier. The user identification handle is contained in the structure pointed to by *ve_user_handle*, from which the channel number can be determined by the function. This structure of type **t_ve_user_handle_s** is described in Appendix B.25 on Page 299.

The direction that the user defined tone is to be played into the audio stream is defined by the *ve_tone_direction* argument. This is an enumerated type, **t_ve_tone_direction_e** as described in Appendix B.79 on Page 518, and allows the generated tone to be played to the local or far end.

Continued >>

A user generated tone can be stopped by calling **VE_GenerateNamedTone** with the VE_TONE_NONE enumerated type. This is the recommended method, although it is also possible to generate another user defined tone with a duration period of zero to stop an user defined tone.

The transmission direction enumerated types can be OR'd together to allow bi-directional transmission of the digit string. Refer to the provided example code of this function to see an instance of this.

This function can only be used if the named_tone_generator_enable element of the t_ve_channel_s structure, described in Appendix B.21 on Page 289, has been enabled.

Example: This example shows different configurations for this function:

```
#include "ve_eapi.c"

{
   t_ve_tone_definition_s  ve_tone;

   /* Define a user tone */
   ve_tone.duration                              = 3000;
   /* Continuous, indicated by -1 */
   ve_tone.repeat_count                          = -1;
   ve_tone.subgroup_count                        = 1;
   ve_tone.subgroup[0].duration                  = 3000;
   ve_tone.subgroup[0].repeat_count              = 0;
   ve_tone.subgroup[0].carrier                   = 0;
   ve_tone.subgroup[0].subtone[0].frequency      = 440;
   ve_tone.subgroup[0].subtone[0].divisor        = 1;
   ve_tone.subgroup[0].subtone[0].scale          = -18;
   ve_tone.subgroup[0].subtone[0].start_time     = 0;
   ve_tone.subgroup[0].subtone[0].end_time       = 3000;
   ve_tone.subgroup[0].subtone[0].on_duration    = 1000;
   ve_tone.subgroup[0].subtone[0].off_duration   = 2000;
   ve_tone.subgroup[0].subtone[0].phase_period   = 0;
   ve_tone.subgroup[0].subtone[1].frequency      = 480;
   ve_tone.subgroup[0].subtone[1].divisor        = 1;
   ve_tone.subgroup[0].subtone[1].scale          = -18;
   ve_tone.subgroup[0].subtone[1].start_time     = 0;
   ve_tone.subgroup[0].subtone[1].end_time       = 3000;
   ve_tone.subgroup[0].subtone[1].on_duration    = 1000;
   ve_tone.subgroup[0].subtone[1].off_duration   = 2000;
   ve_tone.subgroup[0].subtone[1].phase_period   = 0;
```

Continued >>

```
   /* Switch on some state */
   switch(menu_state)
   {
   case MENU_USER_TONE_LOCAL:
      {
   result = VE_GenerateUserTone( applicationId,
                &ve_user_handle,
                &ve_tone,
                VE_TONE_DIRECTION_LOCAL);

   if( result != VE_RETURN_CODE_SUCCESS )
   {
      /* Handle error here */
   }
      }
      break;

   case MENU_USER_TONE_BOTH:
      {
   result = VE_GenerateUserTone(applicationId,
                &ve_user_handle,
                &ve_tone,

(VE_TONE_DIRECTION_LOCAL|VE_TONE_DIRECTION_REMOTE));

   if( result != VE_RETURN_CODE_SUCCESS )
   {
      /* Handle error here */
   }
      }
      break

   case MENU_TONE_NONE_BOTH:
      {
   /* Stop any tones playing on local & remote ends */
   result = VE_GenerateNamedTone( applicationId,
                &ve_user_handle,
                VE_COUNTRY_TYPE_UK,
                VE_TONE_NONE,

(VE_TONE_DIRECTION_LOCAL|VE_TONE_DIRECTION_REMOTE));

   if( result != VE_RETURN_CODE_SUCCESS )
   {
      /* Handle error here */
   }
      }
      break;
```

Continued >>

```
    case MENU_TONE_NONE_LOCAL:
       {
    /* Stop any tones playing on local end */
    result = VE_GenerateNamedTone( applicationId,
                  &ve_user_handle,
                  VE_COUNTRY_TYPE_UK,
                  VE_TONE_NONE,
                  VE_TONE_DIRECTION_LOCAL);

    if( result != VE_RETURN_CODE_SUCCESS )
    {
       /* Handle error here */
    }
       }
       break;

       /* Note: User defined tones can also be stopped by
defining...
    ve_tone.duration = 0;
    ...but recommend the above methods to stop user defined tone */

       /* Other cases can be handled here */

    default:
       {
    /* Handle default case here */
       }
       break;
    }
}
```

VE_GenerateDigitString

Syntax:

```
t_ve_return_code_e  VE_GenerateDigitString(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_signaling_type_e ve_signaling_type,
                    Word8 *ve_digit_string,
                    t_ve_tone_transmission_media_type_e ve_media_type )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user associated with the channel.
ve_signaling_type	Defines the type of signaling to be used.
ve_digit_string	Pointer to a string of digits to generate.
ve_media_type	Defines the method of transmission of the generated digit string.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_GenerateNamedTone**

Description: **VE_GenerateDigitString** is used to generate a specified string of digits. Strings pointed to by *ve_digit_string*, must be NULL terminated. Invalid characters will generate a pause in the digit string.

The *ve_application_id* argument must be a valid registered application identifier. The user identification handle is contained in the structure pointed to by *ve_user_handle*, from which the channel number can be determined by the function. This structure of type **t_ve_user_handle_s** is described in Appendix B.25 on Page 299.

The type of signaling to be used is identified by *ve_signaling_type* and is an enumerated type of **t_ve_signaling_type_e**, as described in Appendix B.76 on Page 515. The transmission media type to be used is identified by *ve_media_type* and is an enumerated type of **t_ve_tone_transmission_media_type_e**, as described in Appendix B.77 on Page 516.

Note that digit strings must be no longer than 30 digits, plus the NULL terminator.

Continued >>

The level, duration and gap duration of the individual digits are determined by the settings within the t_ve_ntg_algorithm_default_configuration_s structure, described in Appendix B.14 on Page 275.

The transmission media types can be OR'd together to allow bi-directional transmission of the digit string. Refer to the provided example code of this function to see an instance of this.

This function can only be used if the `named_tone_generator_enable` element of the t_ve_channel_s structure, described in Appendix B.21.3 on Page 293, has been enabled.

Example: This simple example shows how to use this function:

```
#include "ve_eapi.c"

/* Generate the digit string in both directions */
{
   t_ve_return_code_e result;

   result = VE_GenerateDigitString(applicationId ,
             &user_handle,
             VE_DTMF_SIGNALING,
             "0123456789",
             (VE_TONE_TRANSMISSION_MEDIA_TYPE_AUDIO_LINEAR_PCM|
              VE_TONE_TRANSMISSION_MEDIA_TYPE_PACKET_AUTO_SELECT));

   if(result != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error here */
   }
}
```

Continued >>

Example: This simple example shows another configuration for the **VE_GenerateDigitString** function:

```
#include "ve_eapi.c"

/* Generate the digit string to the audio path only */
{
   t_ve_return_code_e result;

   result = VE_GenerateDigitString(applicationId ,
             &user_handle,
             VE_DTMF_SIGNALING,
             "9876543210",
             VE_TONE_TRANSMISSION_MEDIA_TYPE_AUDIO_LINEAR_PCM);

   if(result != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error here */
   }
}
```

VE_ControlAudio

Syntax:

```
t_ve_return_code_e  VE_ControlAudio(
                      UWord32 ve_application_id,
                      t_ve_user_handle_s *ve_user_handle,
                      t_ve_audio_control_type_e ve_control_type,
                      t_ve_audio_control_u *ve_control_data )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user associated with the channel.
ve_control_type	The type of audio control to be applied to the channel.
Ve_control_data	Pointer to the audio control data structure.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_ControlAudio** is responsible for control of the per channel audio functionality of the VeriCall Edge system and allows the user to play and record audio data.

The *ve_application_id* argument must be a valid registered application identifier. The user identification handle is contained in the structure pointed to by *ve_user_handle*. This structure is of type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299.

The audio control identifier, *ve_control_type*, must be a valid enumeration of type **t_ve_audio_control_type_e**, as described in Appendix B.143 on Page 568.

The control structures encapsulated within the union **t_ve_audio_control_u**, are described in Appendix B.31 on Page 343 and pointed to by *ve_control_data*. The control structures are accessed through the appropriate *ve_control_type* enumeration and where no associated structure encapsulated within *ve_control_data* is present, this entry should be NULL.

Continued >>

Example: This example shows a simple use of this function:

```
#include "ve_eapi.c"

/* Mute the active audio stream */
{
   t_ve_return_code_e result;
   t_ve_audio_control_u audio_config;

   audio_config.ve_audio_mute.state     =
                  VE_AUDIO_CONTROL_DATA_AUDIO_ACTIVATE_MUTE_STATE;
   audio_config.ve_audio_mute.direction =
                  VE_AUDIO_CONTROL_DATA_AUDIO_MUTE_DIRECTION_RX;

   result = VE_ControlAudio( applicationId,
                             &ve_user_handle,
                             VE_AUDIO_CONTROL_TYPE_MUTE_CONFIGURE,
                             &audio_config);

   if(result != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error here */
   }
}
```

Example: This example shows another simple use of this function:

```
#include "ve_eapi.c"

/* Play a mono audio file to the local end */
{
   t_ve_return_code_e result;
   t_ve_audio_control_u audio_config;

   /* Mute the active audio stream */
   audio_config.ve_audio_play_start.ve_format =
                    VE_CONTROL_PLAY_RAW_16BIT_BYTE_REVERSED_MONO;
   audio_config.ve_audio_play_start.ve_direction =
                    VE_CONTROL_DIRECTION_AUDIO;
   strcpy((char*)audio_config.ve_audio_play_start.descriptor,
                                                 "play_file.bin");

   /* Play the file only once */
   result = UA_ControlAudio(applicationId,
                            VE_AUDIO_CONTROL_TYPE_PLAY_ONCE_START,
                            &audio_config);

   if(result != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error here */
   }
}
```

VE_ControlVideo

Syntax:

```
t_ve_return_code_e  VE_ControlVideo(
                      UWord32 ve_application_id,
                      t_ve_user_handle_s *ve_user_handle,
                      t_ve_video_control_type_e ve_control_type,
                      t_ve_video_control_u *ve_control_data )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user associated with the channel.
ve_control_type	The type of video control to be applied to the channel.
ve_control_data	Pointer to the video control data structure.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_ControlVideo** is responsible for control of the per channel video functionality of the VeriCall Edge system and allows the user to manipulate and control the local and received images. This function directly accesses and controls the pre and post-processor of the VeriCall Edge system.

The *ve_application_id* argument must be a valid registered application identifier. The user identification handle is contained in the structure pointed to by *ve_user_handle*. This structure of type **t_ve_user_handle_s** is described in Appendix B.25 on Page 299.

The video control identifier, *ve_control_type*, must be a valid enumeration of type **t_ve_video_control_type_e**, as described in Appendix B.149 on Page 574.

The control structures encapsulated within the union **t_ve_video_control_u**, are described in Appendix B.32 on Page 347 and pointed to by *ve_control_data*. The control structures are accessed through the appropriate *ve_control_type* enumeration and where no associated structure encapsulated within *ve_control_data* is present, this entry should be NULL.

Continued >>

Example: This example shows a simple use of this function:

```
#include "ve_eapi.c"

/* Toggle the local image display state */
{
   t_ve_return_code_e result;
   t_ve_video_control_u param;
   static t_ve_video_control_select_display_state_e display =
                              VE_VIDEO_CONTROL_DISPLAY_STATE_ENABLE;

   if( display == VE_VIDEO_CONTROL_DISPLAY_STATE_ENABLE )
      display = VE_VIDEO_CONTROL_DISPLAY_STATE_DISABLE;
   else
      display = VE_VIDEO_CONTROL_DISPLAY_STATE_ENABLE;

   param.ve_display_state = display;

   result = VE_ControlVideo( applicationId,
           &ve_user_handle,
           VE_VIDEO_CONTROL_TYPE_SELECT_LOCAL_IMAGE_DISPLAY_STATE,
           &param );

   if(result != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error here */
   }
}
```

Example: This example shows another simple use of this function:

```
#include "ve_eapi.c"

/* Capture the received video image */
{
   t_ve_return_code_e result;
   t_ve_video_control_u param;

   param.ve_image_encode.encode_selector =
                              VE_VIDEO_CONTROL_ENCODE_FORMAT_JPEG;
   strcpy((char*)param.ve_image_encode.image, "rx_image.jpg");

   result = VE_ControlVideo( applicationId,
           &user_reg->user_reg[0].ve_user_handle,
           VE_VIDEO_CONTROL_TYPE_ENCODE_RX_IMAGE,
           &param);

   if(result != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error here */
   }
}
```

VE_ControlNetwork

Syntax:

```
t_ve_return_code_e  VE_ControlNetwork(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_network_control_type_e ve_control_type,
                    t_ve_network_control_u *ve_control_data )
```

Arguments:

ve_application_id	The registered application ID.
Ve_user_handle	Pointer to the structure containing the handle of the user associated with the channel.
ve_control_type	The type of network control to be applied.
ve_control_data	Pointer to the network control data structure.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_ControlNetwork** allows for the control of the network parameters, allowing the user application to over-ride the initial the VeriCall Edge system network configuration settings.

The *ve_application_id* argument must be a valid registered application identifier. The user identification handle, used to control the network interface of the channels associated with the user, is contained in the structure pointed to by *ve_user_handle*. This structure of type **t_ve_user_handle_s** is described in Appendix B.25 on Page 299.

The network control identifier, *ve_control_type*, must be a valid enumeration of type **t_ve_network_control_type_e**, as described in Appendix B.127 on Page 554.

The control structures encapsulated within the union **t_ve_network_control_u**, are described in Appendix B.30 on Page 342 and pointed to by *ve_control_data*. The control structures are accessed through the appropriate *ve_control_type* enumeration.

Use of this function ultimately causes an invocation of the user defined Data Manager function registered with VeriCall Edge through the **VE_RegisterPacketTransportConfigure** function.

Continued >>

Example: This simple example shows how to use this function:

```
#include "ve_eapi.h"

{
   t_ve_return_code_e ret_code;
   UWord32   set_tos;
   t_ve_user_handle_s *ve_user_handle

   /* Change network Type of Service (ToS) to be zero (default) */
   set_tos = 0;

   ret_code = VE_ControlNetwork( applicationId,
          &ve_user_handle,
          VE_NETWORK_CONTROL_TYPE_TOS,
          &set_tos );
}
```

VE_NETLookUp

Syntax:

```
t_ve_return_code_e  VE_NETLookUp(
                        UWord32 ve_application_id,
                        t_ve_net_lookup_query_s *lookup_query )
```

Arguments:

ve_application_id	The registered application ID.
lookup_query	Pointer to a structure containing the network service lookup information.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_NETLookUp** provides various network to the VeriCall Edge system and user application. This function allows for the following network services:

- Domain Name Service (DNS) lookup of IPv4 address(es) for a specified host name.
- DNS server (SVR) lookup of server location(s) for specified service type in given domain.
- DNS Naming Authority Pointer (NAPTR) records for a given domain.
- DNS NAPTR lookup for specified E.164 Number Mapping (ENUM).

The ve_application_id argument must be a valid registered application identifier.

The *lookup_query* argument is a structure, **t_ve_net_lookup_query_s** as described in Appendix B.66 on Page 500, that specifies the network lookup service to be performed and the associated lookup details.

Responses to the lookup request are passed to the user application via the VE_USER_MSG_NET_LOOKUP_RESPONSE event that is obtained through use of the **VE_GetNextEvent** functionality. The response event associated with this function call is encapsulated within the **t_ve_user_msg_net_lookup_response_s** structure as described in Appendix B.33.65 on Page 418.

Continued >>

Example: This example shows a simple DNS lookup:

```
#include <ve_net_prototypes.h>

/* Start a DNS lookup to obtain IPv4 address for
   myserver.trinity.local */
{
   t_ve_net_lookup_query_s query;
   t_ve_return_code_e ret_code;

   /* set this to identify query results event */
   query.transaction_id = 0;
   query.type = VE_NET_LOOKUP_TYPE_IPV4_ADDRESS;
   strncpy(query.data.ipv4_address.name, "myserver.trinity.local",
                                         VE_MAX_URI_STRING_SIZE);

   ret_code = VE_NETLookUp( applicationId, &query );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

Example: This example shows a simple ENUM lookup:

```
#include <ve_net_prototypes.h>

/* Start an ENUM lookup to obtain SIP address for telephone number
   * 01234-567-890 using domain e164.arpa */
{
   t_ve_net_lookup_query_s query;
   t_ve_return_code_e ret_code;

   /* set this to identify query results event */
   query.transaction_id = 0;
   query.type = VE_NET_LOOKUP_TYPE_NAPTR_ENUM;
   query.data.naptr_enum.service_type = VE_NET_SERVICE_TYPE_SIP;
   strncpy(query.data.naptr_enum.phone_number, "01234-567-890",
               user_regs->user_reg[user_index].num_rx_dtmf_digits);
   strcpy(query.data.naptr_enum.domain, "e164.arpa");

   ret_code = VE_NETLookUp( application_id, &query );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

Chapter 6

VeriCall Edge™ Data Manager API

This chapter describes the VeriCall Edge Data Manager API registration functions. The VeriCall Edge Data Manager API is used to register the application Data Manager interface functions with the VeriCall Edge system. Refer to the VeriCall Edge User Manual (MAN20003) for information on the Data Manager functions to be registered.

6.1 VeriCall Edge Data Manager API Function Descriptions

For each function detailed in this section, the following are described (as appropriate):

- **Syntax** Command syntax.
- **Arguments** Description of the arguments.
- **Returns** Return codes from the function.
- **See Also** Other functions that perform related tasks.
- **Description** What the function can be used for.
- **Example** Short code sections showing how to use the function.

6.2 VeriCall Edge Data Manager Functions

The VeriCall Edge Data Manager functions are described on the following pages:

Function	Page
VE_RegisterPacketTransportOpen	79
VE_RegisterPacketTransportSend	81
VE_RegisterPacketTransportReceive	83
VE_RegisterPacketTransportConfigure	85
VE_RegisterPacketTransportClose	87
VE_RegisterChannelDataSourceSinkOpen	89
VE_RegisterChannelDataTransfer	91
VE_RegisterChannelDataSourceSinkClose	92
VE_RegisterChannelDataTransferAvailability	95
VE_RegisterChannelDataDeviceQuery	97
VE_RegisterChannelDataDeviceConfigure	100

VE_RegisterPacketTransportOpen

Syntax:

```
t_ve_return_code_e  VE_RegisterPacketTransportOpen(
                t_ve_user_callback_return_code_e *fn_ptr(
                        t_ve_callback_network_open_s *configuration,
                        t_ve_callback_host_descriptor_s *host,
                        UWord16 port,
                        UWord32 *handle ))
```

Arguments:

fn_ptr	Pointer to the open transport function to register.
configuration	Pointer to a structure containing protocol to be used and the direction of the packet data.
host	Pointer to the host name to which the connection should be made.
port	The port number to which the connection should be made.
handle	A unique handle returned by the VeriCall Edge system to identify the connection in future transactions.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_RegisterPacketTransportClose**

Description: **VE_RegisterPacketTransportOpen** is used to register the Open Transport callback function with the VeriCall Edge system user application. This callback function is used by the VeriCall Edge system to open a packet-side connection.

The registered callback function is defined via the *fn_ptr* pointer and will return an enumerated code indicating function registration success or failure. It is expected that the user supplied function will return values of the type **t_ve_user_callback_return_code_e** enumerated type is described in Appendix B.169 on Page 594.

The registered function must present arguments as described above.

The *configuration* argument is a pointer to the **t_ve_callback_network_open_s** structure, described in Appendix B.23 on Page 297, allows for the protocol and transport direction to be associated with this registered user function.

Continued >>

The *host* argument is a pointer to a **t_ve_callback_host_descriptor_s** structure type described in Appendix B.35 on Page 425 and determines the host name of the connection to be opened. The *port* argument is a suitable port number to either listen on (for receive connections) or to transmit to (for transmit connections). Each opened packet transport connection is given a unique identifying handle which is placed into the area of memory pointed to by *handle*. This handle is used within the other packet-side callback functions to identify this specific connection.

Example: This example shows how register the user written function:

```
#include "ve_eapi.h"

/* Register the call-back function for the IP transport open
   functionality */
t_ve_user_callback_return_code_e UA_DM_OpenIPTransport
 (
    t_ve_callback_network_open_s      *configuration, /* input */
    t_ve_callback_host_descriptor_s   *host,          /* input */
    UWord16                           port,           /* input */
    UWord32                           *handle         /* output */
   )
{
   /* Open the IP transport port and return a unique handle */
   return VE_USER_CALLBACK_SUCCESS;
}

{
   t_ve_return_code_e ret_code;

   ret_code =
VE_RegisterPacketTransportOpen(UA_DM_OpenIPTransport);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_RegisterPacketTransportSend

Syntax:

```
t_ve_return_code_e  VE_RegisterPacketTransportSend(
                t_ve_user_callback_return_code_e *fn_ptr(
                        UWord32 handle,
                        t_ve_callback_host_descriptor_s *host,
                        UWord16 tx_port,
                        UWord8 *message,
                        UWord32 message_size ))
```

Arguments:

fn_ptr	Pointer to the user callback function.
handle	The handle that is returned by the registered function to the VeriCall Edge system.
host	Pointer to the host name to which the connection should be made.
tx_port	The transmit port number to which the connection should be made.
message	Pointer to the message to be sent across the network connection.
message_size	The size of the message to be sent across the network connection.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_RegisterPacketTransportReceive**

Description: **VE_RegisterPacketTransportSend** is used to register the Send Transport callback function with the VeriCall Edge system user application. This callback function is used by the VeriCall Edge system to send data to an open packet-side connection.

The registered callback function is defined via the *fn_ptr* pointer and will return an enumerated code indicating function registration success or failure. It is expected that the user-supplied function will return values of the enumerated type **t_ve_user_callback_return_code_e** as described in Appendix B.169 on Page 594.

The registered function must present arguments as described above. The *handle* argument references the handle returned from the **VE_RegisterPacketTransportOpen** function and indicates that the data must be transmitted through this specific connection.

The *host* argument is a pointer to a **t_ve_callback_host_descriptor_s** structure type described in Appendix B.35 on Page 425 and determines the host name to which the data should be sent. The *tx_port* argument is the port number to which the data should be transmitted via.

Continued >>

The message to be sent via the callback function registered with this function is pointed to by *message* argument while the actual size of the message to be sent is determined by the *message_size* argument.

Example: This example shows how register the user written function:

```
#include "ve_eapi.h"

/* Register the call-back function for the IP transport send
   functionality */
t_ve_user_callback_return_code_e UA_DM_IPTransportSend
   (
    UWord32                              handle,          /* input
*/
    t_ve_callback_host_descriptor_s      *host,          /* input */
    UWord16                              tx_port,        /* input */
    UWord8                               *message,       /* input */
    UWord32                              message_size    /* input */
   )
{
   /* Send the given message on the port associated with the given
      handle */
   return VE_USER_CALLBACK_SUCCESS;
}

{
   t_ve_return_code_e ret_code;

   ret_code =
VE_RegisterPacketTransportSend(UA_DM_IPTransportSend);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_RegisterPacketTransportReceive

Syntax:

```
t_ve_return_code_e  VE_RegisterPacketTransportReceive(
                t_ve_user_callback_return_code_e fn_ptr(
                        UWord32 *handle,
                        UWord8 *message,
                        UWord32 message_size_max,
                        UWord32 *message_size,
                        t_ve_callback_host_descriptor *rx_host,
                        UWord16 *rx_port ))
```

Arguments:

fn_ptr	Pointer to the user callback function.
handle	The handle that was returned to the VeriCall Edge system when the transport connection was opened.
message	Pointer to storage for the message to be received.
message_size_max	The maximum size of message to be received.
message_size	The size of the message to be received.
rx_host	Pointer to the host name from which the data is to be received.
rx_port	Pointer to the receive port number from which data is to be received.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_RegisterPacketTransportSend**

Description: **VE_RegisterPacketTransportReceive** is used to register the Transport Receive callback function with the VeriCall Edge system user application. This callback function is used by the VeriCall Edge system to receive data from an open packet-side connection.

The registered callback function is defined via the *fn_ptr* pointer and will return an enumerated code indicating function registration success or failure. It is expected that the user-supplied function will return values of the enumerated type **t_ve_user_callback_return_code_e** as described in Appendix B.169 on Page 594.

The registered function must present arguments as described above. The *handle* argument references the handle returned from the **VE_RegisterPacketTransportOpen** function and indicates that the data must be received through this specific connection.

Continued >>

The handle argument is a pointer to support TCP/IP for the NAT/SIP modules. When a TCP/IP connection is opened, a connection is "listened" for on handle *x*. When the connection is established, a new handle *y* is obtained which then needs to be passed to the caller.

The message to be received is stored in an area of memory pointed to by the *message* argument while the actual size of the message to be received is determined by the *message_size* argument. The maximum size of message that can be received is indicated by the *message_size* argument and will be dependent on the transport protocol used.

The *rx_host* argument is a pointer to a **t_ve_callback_host_descriptor_s** structure type described in Appendix B.35 on Page 425 and determines the host name from which the data is to be received. The *rx_port* argument is a pointer to the port number from which the data is to be retrieved.

Example: This example shows how register the user written function:

```
#include "ve_eapi.h"

/* Register the call-back function for the IP transport receive
   functionality */
t_ve_user_callback_return_code_e UA_DM_IPTransportReceive
(
   UWord32                          *handle,          /* in/out */
   UWord8                           *message,         /* input */
   UWord32                          message_size_max, /* input */
   UWord32                          *message_size,     /* output */
   t_ve_callback_host_descriptor_s  *rx_host,         /* output */
   UWord16                          *rx_port          /* output */
)
{
   /* Attempt to receive a packet on the port associated with the
      given handle */
   return VE_USER_CALLBACK_SUCCESS;
}

{
   t_ve_return_code_e ret_code;

   ret_code = VE_RegisterPacketTransportReceive(
                                       UA_DM_IPTransportReceive );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_RegisterPacketTransportConfigure

Syntax:
```
t_ve_return_code_e  VE_RegisterPacketTransportConfigure(
                t_ve_user_callback_return_code_e *fn_ptr(
                        UWord32 handle,
                        t_ve_callback_network_config_s *configure ))
```

Arguments:

fn_ptr	Pointer to the user callback function.
handle	The handle that was returned to the VeriCall Edge system when the transport connection was opened.
configure	Pointer to the structure containing the network configuration parameters.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_RegisterPacketTransportSend**, **VE_ControlNetwork**

Description: **VE_RegisterPacketTransportConfigure** is used to register the Transport Configure function with the VeriCall Edge system user application. This callback function is used by the VeriCall Edge system to adjust the network interface parameters from the User Application.

The registered callback function is defined via the *fn_ptr* pointer and will return an enumerated code indicating function registration success or failure. It is expected that the user-supplied function will return values of the enumerated type **t_ve_user_callback_return_code_e** as described in Appendix B.169 on Page 594.

The registered function must present arguments as described above. The *handle* argument references the handle returned from the **VE_RegisterPacketTransportOpen** function and indicates the connection to which the configuration is to be applied to.

The *configure* argument is a pointer to a structure of type **t_ve_callback_network_config_s** as described in Appendix B.37 on Page 430. The structure provides the settings and values for the network parameters to be configured.

The function registered through this callback is used in conjunction with the VE_ControlNetwork function.

Continued >>

Example: This example shows how register the user written function:

```
#include "ve_eapi.h"

/* Register the call-back function for the IP transport configure
   functionality */
t_ve_user_callback_return_code_e UA_DM_IPTransportConfigure
(
    UWord32                          handle,
    t_ve_callback_network_config_s  *configuration
)
{
   /* Configure the port associated with the given handle */
   return VE_USER_CALLBACK_SUCCESS;
}

{
   t_ve_return_code_e ret_code;

   ret_code = VE_RegisterPacketTransportConfigure(
                                    UA_DM_IPTransportConfigure );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_RegisterPacketTransportClose

Syntax:

```
t_ve_return_code_e  VE_RegisterPacketTransportClose(
                t_ve_user_callback_return_code_e *fn_ptr(
                        UWord32 handle ))
```

Arguments:

fn_ptr	Pointer to the user callback function.
handle	The handle that was returned by the user registered function.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_RegisterPacketTransportOpen**

Description: **VE_RegisterPacketTransportClose** is used to register the Close Transport callback function with the VeriCall Edge system user application. This callback function is used by the VeriCall Edge system to close a previously opened packet-side connection.

The registered callback function is defined via the *fn_ptr* pointer and will return an enumerated code indicating function registration success or failure. It is expected that the user-supplied function will return values of the enumerated type **t_ve_user_callback_return_code_e** as described in Appendix B.169 on Page 594.

The registered function must present arguments as described above. The *handle* argument references the handle returned from the **VE_RegisterPacketTransportOpen** function and indicates the specific connection which to be closed.

Continued >>

Example: This example shows how register the user written function:

```
#include "ve_eapi.h"

/* Register the call-back function for the IP transport close
   functionality */
t_ve_user_callback_return_code_e UA_DM_CloseIPTransport
(
    UWord32    handle                                    /* input */
)
{
  /* Close the IP transport port associated with given handle */
   return VE_USER_CALLBACK_SUCCESS;
}

{
   t_ve_return_code_e ret_code;

   ret_code = VE_RegisterPacketTransportClose(
                                          UA_DM_CloseIPTransport );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_RegisterChannelDataSourceSinkOpen

Syntax:

```
t_ve_return_code_e  VE_RegisterChannelDataSourceSinkOpen(
  UWord32 identifier,
  t_ve_user_callback_return_code_e *fn_ptr(
       t_ve_callback_direction_e direction,
       t_ve_callback_data_channel_descriptor_s *dev_descriptor,
       UWord32 data_end_point_instance,
       UWord32 *handle ))
```

Arguments:

identifier	Label to identify this specific callback entry within the VeriCall Edge system.
fn_ptr	Pointer to the user callback function.
direction	Enumerated type defining the direction of the callback function data.
dev_descriptor	Pointer to the descriptor of the device driver to use for data transfer.
data_end_point_instance	The associated data-end point instance of this data source/sink open function.
handle	A unique handle returned to the VeriCall Edge system to identify the connection in future transactions.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_RegisterChannelDataSourceSinkClose**

Description: **VE_RegisterChannelDataSourceSinkOpen** is used to register an Open Data connection callback function with the VeriCall Edge system user application. This callback function is used by the VeriCall Edge system to open a per-channel data-side connection to an identified data-end point instance.

For multiple data-end point instances, the vendor written function/functions associated to each data-end point should be registered with the channel. The *identifier* argument is a label used to identify this callback function within the VeriCall Edge system and should match the identified data-end point instance described within the **t_ve_channel_data_configuration_s** structure as described in Appendix B.21.2 on Page 291.

Continued >>

The registered callback function/functions are defined via the *fn_ptr* pointer and will return an enumerated code indicating function registration success or failure. It is expected that the user-supplied function will return values of the enumerated type **t_ve_user_callback_return_code_e** as described in Appendix B.169 on Page 594.

The registered function must present the arguments as described above. The **t_ve_callback_direction_e** enumerated type defines the direction of the data associated with this callback function, and is described in Appendix B.172 on Page 596. The **t_ve_callback_data_channel_descriptor_s** structure is described in Appendix B.34 on Page 424. The *data_end_point_instance* argument indicates the data-end point instance that is to be opened. The *handle* argument is a pointer to storage for a unique identifier returned to the VeriCall Edge system to be used in future associated data transactions.

Example: This example shows how register the user written function:

```
#include "ve_eapi.h"

/* Register the call-back function for the audio interface open
   functionality */
t_ve_user_callback_return_code_e UA_DM_OpenAudio
(
    t_ve_callback_direction_e                  direction,
    t_ve_callback_data_channel_descriptor_s    *dev_descriptor,
    UWord32                                data_end_point_instance,
    UWord32                                *handle
)
{
   /* Open the requested audio interface for the given
      data_end_point_instance and direction then return a unique
      handle */
   return VE_USER_CALLBACK_SUCCESS;
}

{
   t_ve_return_code_e ret_code;

   ret_code = VE_RegisterChannelDataSourceSinkOpen(
                                                UA_DM_OpenAudio );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_RegisterChannelDataTransfer

Syntax:

```
t_ve_return_code_e  VE_RegisterChannelDataTransfer(
                UWord32 identifier,
                t_ve_user_callback_return_code_e *fn_ptr(
                        UWord32 handle,
                        UWord32 data_end_point_instance,
                        UWord8 *data_buffer,
                        UWord32 *size ))
```

Arguments:

identifier	Label to identify this specific callback entry within the VeriCall Edge system.
fn_ptr	Pointer to the user callback function.
handle	The handle that was returned to the VeriCall Edge system when the connection was opened.
data_end_point_instance	The associated data-end point instance of this data transfer.
data_buffer	Pointer to the data to be transferred to/from the VeriCall Edge system.
size	The size of the data to be transferred.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_RegisterChannelDataTransferAvailability**

Description: **VE_RegisterChannelDataTransfer** is used to register either a Send Data or Receive Data callback function with the VeriCall Edge system user application. This callback function is used by the VeriCall Edge system to either send or receive data to a per-channel data-side connection.

For multiple data-end point instances, the vendor written function/functions should be registered multiple times per-channel. The *identifier* argument is a label used to identify this callback function within the VeriCall Edge system and should match the identified data-end point instance described within the **t_ve_channel_data_configuration_s** structure as described in Appendix B.21.2 on Page 291.

The registered callback function/functions are defined via the *fn_ptr* pointer and will return an enumerated code indicating function registration success or failure. It is expected that the user-supplied function will return values of the enumerated type **t_ve_user_callback_return_code_e** as described in Appendix B.169 on Page 594.

Continued >>

The registered function must present the arguments as described above. The *handle* argument should be that returned from the function registered via **VE_RegisterChannelDataSourceSinkOpen**, for the specific channel and data-end point instance. The *data_end_point_instance* argument indicates the data-end point instance that the send/receive data transfer is to be performed on.

The location and size of the data to be transferred (sent or received) are pointed to by the *data_buffer* and *size* arguments.

Example: This example shows how register the user written function:

```
#include "ve_eapi.h"

/* Register the call-back function for the audio interface data
   transfer functionality */
t_ve_user_callback_return_code_e UA_DM_AudioSendReceive
 (
    UWord32                    handle,
    UWord32                    data_end_point_instance,
    UWord8                     *data_buffer,
    UWord32                    *size
)
{
   /* Depending on the directionality associated with the given
      handle transfer data to or from the audio device */
   return VE_USER_CALLBACK_SUCCESS;
}

{
   t_ve_return_code_e ret_code;

   ret_code = VE_RegisterChannelDataTransfer(
                                          UA_DM_AudioSendReceive );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_RegisterChannelDataSourceSinkClose

Syntax:

```
t_ve_return_code_e  VE_RegisterChannelDataSourceSinkClose(
                UWord32 identifier,
                t_ve_user_callback_return_code_e *fn_ptr(
                        UWord32 handle
                        UWord32 data_end_point_instance ))
```

Arguments:

identifier	Label to identify this specific callback entry within the VeriCall Edge system.
fn_ptr	Pointer to the user callback function.
handle	The handle that was returned to the VeriCall Edge system when the connection was opened.
data_end_point_instance	The associated data-end point instance of the source being closed.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_RegisterChannelDataSourceSinkOpen**

Description: **VE_RegisterChannelDataSourceSinkClose** is used to register the Close Data callback function with the VeriCall Edge system user application. This callback function is used by the VeriCall Edge system to close a previously opened per-channel data-side connection to an identified data-end point instance.

For multiple data-end point instances, the vendor written function/functions should be registered multiple times per-channel. The *identifier* argument is a label used to identify this callback function within the VeriCall Edge system and should match the identified data-end point instance described within the **t_ve_channel_data_configuration_s** structure as described in Appendix B.21.2 on Page 291.

The registered callback function/functions are defined via the *fn_ptr* pointer and will return an enumerated code indicating function registration success or failure. It is expected that the user-supplied function will return values of the enumerated type **t_ve_user_callback_return_code_e** as described in Appendix B.169 on Page 594.

The registered function must present the arguments as described above. The *handle* argument should be that returned from the function registered via **VE_RegisterChannelDataSourceSinkOpen**, for the specific channel and data-end point instance. The *data_end_point_instance* argument indicates the data-end point instance that is to be closed.

Continued >>

Example: This example shows how register the user written function:

```
#include "ve_eapi.h"

/* Register the call-back function for the audio interface close
   functionality */
t_ve_user_callback_return_code_e UA_DM_CloseAudio
(
    UWord32                   handle,
    UWord32                   data_end_point_instance
)
{
   /* Close the audio interface associated with the given
      end-point instance and handle */
   return VE_USER_CALLBACK_SUCCESS;
}

{
   t_ve_return_code_e ret_code;

   ret_code = VE_RegisterChannelDataSourceSinkClose(
                                                  UA_DM_CloseAudio );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_RegisterChannelDataTransferAvailability

Syntax:

```
t_ve_return_code_e  VE_RegisterChannelDataTransferAvailability(
                UWord32 identifier,
                t_ve_user_callback_return_code_e *fn_ptr(
                        UWord32 handle,
                        UWord32 data_end_point_instance
                        UWord32 *tx_available ))
```

Arguments:

identifier	Label to identify this specific callback entry within the VeriCall Edge system.
fn_ptr	Pointer to the user callback function.
handle	The handle that was returned to the VeriCall Edge system when the connection was opened.
data_end_point_instance	The data-end point instance of the data transferal.
tx_available	The amount of transfer space available to send/receive the data.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_RegisterChannelDataTransfer**

Description: **VE_RegisterChannelDataTransferAvailability** is used to register either a Send Data Space Available or Receive Data Space Available callback function with the VeriCall Edge system user application. This callback function is used by the VeriCall Edge system to determine the amount of space available, dependent on whether the function is associated with a sink or a source, on a per-channel data-side connection for an identified data-end point instance.

For multiple data-end point instances, the vendor written function/functions should be registered multiple times per-channel. The *identifier* argument is a label used to identify this callback function within the VeriCall Edge system and should match the identified data-end point instance described within the **t_ve_channel_data_configuration_s** structure as described in Appendix B.21.2 on Page 291.

The registered callback function/functions are defined via the *fn_ptr* pointer and will return an enumerated code indicating function registration success or failure. It is expected that the user-supplied function will return values of the enumerated type **t_ve_user_callback_return_code_e** as described in Appendix B.169 on Page 594.

Continued >>

The registered function must present the arguments as described above. The *handle* argument should be that returned from the function registered via **VE_RegisterChannelDataSourceSinkOpen**, for the specific channel and data-end point instance. The *data_end_point_instance* argument indicates the data-end point instance that is to be closed.

The amount of space available for sending/receiving is pointed to by the *tx_available* argument and is measured in bytes.

Example: This example shows how register the user written function:

```
#include "ve_eapi.h"

/* Register the call-back function for the audio interface data
   availability functionality */
t_ve_user_callback_return_code_e UA_DM_AudioSendSpaceDataAvailable
(
    UWord32                  handle,
    UWord32                  data_end_point_instance,
    UWord32                  *space_available
)
{
   /* Depending on the directionality associated with the given
      handle determine the data available from the audio device or
      the space available on the audio device */
   return VE_USER_CALLBACK_SUCCESS;
}

{
   t_ve_return_code_e ret_code;

   ret_code = VE_RegisterChannelDataTransferAvailability(
                               UA_DM_AudioSendSpaceDataAvailable );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_RegisterChannelDataDeviceQuery

Syntax:

```
t_ve_return_code_e  VE_RegisterChannelDataDeviceQuery(
          UWord32 identifier,
          t_ve_user_callback_return_code_e *fn_ptr(
                UWord32 handle,
                UWord32 data_end_point_instance
                t_ve_callback_device_query_u *device_config ))
```

Arguments:

identifier	Label to identify this specific callback entry within the VeriCall Edge system.
fn_ptr	Pointer to the user callback function.
handle	The handle that was returned to the VeriCall Edge system when the data connection was opened.
data_end_point_instance	The associated data-end point instance of this device query function.
device_config	Pointer to union of structures to encapsulate the information retrieved.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_RegisterChannelDataDeviceConfigure**

Description: **VE_RegisterChannelDataDeviceQuery** is used to register a Device Query callback function with the VeriCall Edge system user application. This callback function is used by the VeriCall Edge system to query the underlying per-channel identified data-end point instance hardware or system, returning the required information to the VeriCall Edge system.

The registered function/functions can be used for either audio or video queries. When the function is associated to an audio channel, then the function is responsible for retrieving the following parameters:

- Sample rate
- Buffer Depth
- Duplex Capabilities
- Maximum Latency

Continued >>

When the function is associated to a video channel, the function is responsible for retrieving the following parameters:

- Frame rate
- Frame buffer address
- Image width
- Image height
- Transparency
- Buffer type – Static or virtual
- Transfer mode – Polled or Interrupt
- Color format – YUV420/422 and RGB 16/32- bit

For multiple data-end point instances, the vendor written function/functions should be registered multiple times per-channel. The *identifier* argument is a label used to identify this callback function within the VeriCall Edge system and should match the identified data-end point instance described within the **t_ve_channel_data_configuration_s** structure as described in Appendix B.21.2 on Page 291.

The registered callback function/functions are defined via the *fn_ptr* pointer and will return an enumerated code indicating function registration success or failure. It is expected that the user-supplied function will return values of the enumerated type **t_ve_user_callback_return_code_e** as described in Appendix B.169 on Page 594.

The parameters that the registered function/functions should return to the VeriCall Edge system are defined, and should stored within, the structures encapsulated within the **t_ve_callback_device_query_u** union, as defined in Appendix B.36 on Page 426.

Example: This example shows how register the user written function:

```
#include "ve_eapi.h"

/* Register the call-back function for the audio interface query
   functionality */
t_ve_user_callback_return_code_e UA_DM_AudioDeviceQuery
(
    UWord32                          handle,
    UWord32                          data_end_point_instance,
    t_ve_callback_device_query_u     *device_config
)
{
   /* Determine the configuration of the audio device associated
      with the given handle and return the details in the
      device_config structure */
   return VE_USER_CALLBACK_SUCCESS;
}
```

Continued >>

```
{
   t_ve_return_code_e ret_code;

   ret_code = VE_RegisterChannelDataDeviceQuery(
                                          UA_DM_AudioDeviceQuery );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_RegisterChannelDataDeviceConfigure

Syntax:

```
t_ve_return_code_e  VE_RegisterChannelDataDeviceConfigure(
          UWord32 identifier,
          t_ve_user_callback_return_code_e *fn_ptr(
               UWord32 data_end_point_instance
               UWord32 handle,
               t_ve_callback_data_channel_media_format_e format,
               t_ve_callback_data_channel_config_u *dev_config ))
```

Arguments:

identifier	Label to identify this specific callback entry within the VeriCall Edge system.
fn_ptr	Pointer to the user callback function.
data_end_point_instance	The data-end point instance that this configuration is to be associated with.
handle	The handle that was returned to the VeriCall Edge system when the connection was opened.
format	Defines the media data-format to be used.
dev_config	Defines the type of data to be transferred through the callback function.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_RegisterChannelDataDeviceQuery**

Description: **VE_RegisterChannelDataDeviceConfigure** is used to register a Device Configure callback function with the VeriCall Edge system user application. This callback function is used by the VeriCall Edge system to configure per-channel identified data-end point instance.

The registered function can be used for either audio or video device configurations. When the function is associated to an audio channel, then the function is responsible for configuring the following device parameters:

- Sample rate
- Buffer Depth
- Big or little endian use

Continued >>

When the function is associated to a video channel, the function is responsible for configuring the following device parameters:

- Format – YUV420/422 and RGB 16/32-bit
- Frame rate
- Width
- Height

For multiple data-end point instances, the vendor written function/functions should be registered multiple times per-channel. The *identifier* argument is a label used to identify this callback function within the VeriCall Edge system and should match the identified data-end point instance described within the **t_ve_channel_data_configuration_s** structure as described in Appendix B.21.2 on Page 291.

The registered callback function/functions are defined via the *fn_ptr* pointer and will return an enumerated code indicating function registration success or failure. It is expected that the user-supplied function will return values of the enumerated type **t_ve_user_callback_return_code_e** as described in Appendix B.169 on Page 594.

The format of the media data expected to be received by the VeriCall Edge system is defined by the *format* argument. This enumeration is of type **t_ve_callback_data_channel_media_format_e** as defined in Appendix B.174 on Page 597. The configuration parameters that the registered function/functions should accept from the VeriCall Edge system are defined in the structures encapsulated within the **t_ve_callback_channel_data_configuration_u** union, as defined in Appendix B.22 on Page 295.

Example: This example shows how register the user written function:

```
#include "ve_eapi.h"

/* Register the call-back function for the audio interface
   configure functionality */
t_ve_user_callback_return_code_e UA_DM_ConfigureAudio
(
    UWord32                                 data_end_point_instance,
    UWord32                                       handle,
    t_ve_callback_data_channel_media_format_e  format,
    t_ve_callback_data_channel_config_u        *dev_config
)
{
   /* Configure the audio interface associated with the given
      end-point instance and handle with the given configuration */
   return VE_USER_CALLBACK_SUCCESS;
}
```

Continued >>

```
{
   t_ve_return_code_e ret_code;

   ret_code = VE_RegisterChannelDataDeviceConfigure(
                                                UA_DM_ConfigureAudio );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

Chapter 7

VeriCall Edge™ SIP API

This chapter describes the VeriCall Edge SIP API functions. The VeriCall Edge SIP API is used for call handling.

7.1 VeriCall Edge SIP API Function Descriptions

For each function detailed in this section, the following are described (as appropriate):

- **Syntax** Command syntax.
- **Arguments** Description of the arguments.
- **Returns** Return codes from the function.
- **See Also** Other functions that perform related tasks.
- **Description** What the function can be used for.
- **Example** Short code sections showing how to use the function.

7.2 VeriCall Edge SIP API Functions

The VeriCall Edge SIP API functions are described on the following pages:

Function	Page
VE_SIPRegisterUser	106
VE_SIPDeregisterUser	108
VE_SIPAddUsersRegistrationData	110
VE_SIPGetUsersRegistrationData	112
VE_SIPUpdateUsersRegistrationData	114
VE_SIPRemoveUsersRegistrationData	116
VE_SIPUpdateRegistration	118
VE_SIPQueryRegistration	120
VE_SIPSessionStart	122
VE_SIPSessionStart_WithConfig	124
VE_SIPSessionStop	127
VE_SIPSessionResponse	129
VE_SIPSessionUpdate	131
VE_SIPSessionReinvite	134
VE_SIPSessionHoldOn	136
VE_SIPSessionHoldOff	138
VE_SIPSessionTransfer	140
VE_SIPSessionTransferConsult	143
VE_SIPSessionTransferDontConsult	145
VE_SIPSendMessage	147
VE_SIPSendOptions	150
VE_SIPSendInfo	153
VE_SIPSendService	156
VE_SIPAddUsersPresenceData	158
VE_SIPGetUsersPresenceData	160
VE_SIPUpdateUsersPresenceData	162
VE_SIPRemoveUsersPresenceData	164
VE_SIPStartPresenceSubscription	165
VE_SIPStopPresenceSubscription	167
VE_SIPStartMessageWaitingSubscription	169

VE_SIPStopMessageWaitingSubscription 171
VE_SIPInterrogateMessageWaiting 173
VE_SIPStartDialogSubscription 175
VE_SIPStopDialogSubscription 177
VE_SIPInterrogateDialogSubscription 179

VE_SIPRegisterUser

Syntax:

```
t_ve_return_code_e  VE_SIPRegisterUser(
                        UWord32 ve_application_id,
                        t_ve_user_handle_s *ve_user_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the previously added user to the VeriCall Edge system.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_AddUser**, **VE_RemoveUser**, **VE_SIPDeregisterUser**, **VE_SIPAddUsersRegistrationData**

Description: **VE_SIPRegisterUser** is used to register a user with a SIP registration server. Before this function is used, a user must have been added to the VeriCall Edge system, through the **VE_AddUser** function, and a call to **VE_SIPAddUsersRegistrationData** must have been made, which contains the users SIP registration details.

The *ve_application_id* argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

Example: This example shows how to register a user with the VeriCall Edge system:

```
#include <ve_sip_prototypes.h>

/* Start a new SIP registration */
{
   t_ve_return_code_e           ret_code;
   t_ve_sip_user_registration_s ve_user_reg;

   strncpy( (char*)ve_user_reg.sip_user_uri.uri,
            "sip:user_1@trinity.com", VE_MAX_URI_STRING_SIZE);

   strncpy( (char*)ve_user_reg.sip_reg_server.password,
            (char*)"Password", VE_MAX_PASSWORD_SIZE);
```

Continued >>

```
    /* Add the user registration data into the data store */
    ret_code = VE_SIPAddUsersRegistrationData(applicationId,
                    &user_handle,
                    &ve_user_reg);

    if(ret_code != VE_RETURN_CODE_SUCCESS)
    {
       /* Handle error */
    }

    /* Initiate the SIP registration to the registration server */
    ret_code = VE_SIPRegisterUser( applicationId,
           &ve_user_handle );
    if(ret_code != VE_RETURN_CODE_SUCCESS)
    {
       /* Handle error */
    }
    /* Expect a success or failure event */
}
```

VE_SIPDeregisterUser

Syntax:

```
t_ve_return_code_e  VE_SIPDeregisterUser(
UWord32 ve_application_id,
t_ve_sip_deregister_contacts_flag_e ve_deregister_contacts_flag,
t_ve_user_handle_s *ve_user_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_deregister_contacts_flag	Enumerated type that defines how deregistration is to be performed.
ve_user_handle	Pointer to structure containing the handle of the user, previously added to the VeriCall Edge system, that is to be removed.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_RemoveUser**, **VE_SIPRegisterUser**

Description: **VE_SIPDeregisterUser** is used to de-register a user from a SIP registration server. This function only de-registers the user from the SIP registration server, the user is still registered with the VeriCall Edge system.

The ve_application_id argument must be a valid registered application identifier.

The *ve_deregister_contacts_flag* enumerated type, **t_ve_sip_deregister_contacts_flag_e** as described in Appendix B.184 on Page 606, is used to determine how the deregistration process is to be performed. This enumerated type allows the deregistration of only the local contacts created for the identified user or all the contacts at the registration server, including those not created for the identified user.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call and is used to identify the user that is to be deregistered.

Continued >>

Example: This example shows how to deregister a user from the VeriCall Edge system:

```
#include <ve_sip_prototypes.h>

/* If the user logs out of the application, the application must
   deregister the user and all f their contacts */
{
   t_ve_return_code_e           ret_code;

   /* Initiate the SIP deregistration asking the registration
      server to expire ALL the contacts listed in the
      UsersRegistrationData. NOTE: The application could use the
      VE_SIP_DEREGISTER_ALL_CONTACTS this will deregister ALL
      contacts the registration server associates with this
      user. Be careful though, because if the user has two phones,
      one a desk phone another portable and they log out of their
      desk phone with VE_SIP_DEREGISTER_ALL_CONTACTS it will cause
      their portable to deregister also! */

   ret_code = VE_SIPDeregisterUser(applicationId,
             VE_SIP_DEREGISTER_LOCAL_CONTACTS,
             &ve_user_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect a success or failure event */

   /* Wait until a success event is seen */

   /* Clear the registration data from the data store */
   ret_code = VE_SIPRemoveUsersRegistrationData(applicationId,
                                                &user_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* The application is now free for someone else to login */
}
```

VE_SIPAddUsersRegistrationData

Syntax:

```
t_ve_return_code_e  VE_SIPAddUsersRegistrationData(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_sip_user_registration_s *user_registration )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the added user.
user_registration	Pointer to the structure containing the users registration details.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_AddUser**, **VE_SIPRegisterUser**, **VE_SIPRemoveUsersRegistrationData**

Description: **VE_SIPAddUsersRegistrationData** is used to add user SIP registration details required by the VeriCall Edge system. The user must have been previously added to the VeriCall Edge system through the **VE_AddUser** function. These user SIP registration details will also be used if the user is to be registered against a SIP registration server, performed by using the **VE_SIPRegisterUser** call.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The users SIP registration information is held in the structure pointed to by *user_registration*. This structure of type **t_ve_sip_user_registration_s**, is described in Appendix B.39 on Page 440.

Continued >>

Example: This example shows how to add a users registration data:

```
#include <ve_sip_prototypes.h>

/* Start a new SIP registration */
{
   t_ve_return_code_e           ret_code;
   t_ve_sip_user_registration_s ve_user_reg;

   strncpy( (char*)ve_user_reg.sip_user_uri.uri,
     "sip:user_1@trinity.com", VE_MAX_URI_STRING_SIZE);
   strncpy( (char*)ve_user_reg.sip_reg_server.password,
     (char*)"Password", VE_MAX_PASSWORD_SIZE);

   /* Add the user registration data into the data store */
   ret_code = VE_SIPAddUsersRegistrationData(applicationId,
                  &user_handle,
                  &ve_user_reg);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Initiate the SIP registration to the registration server */
   ret_code = VE_SIPRegisterUser( applicationId,
           &ve_user_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect a success or failure event */
}
```

VE_SIPGetUsersRegistrationData

Syntax:

```
t_ve_return_code_e  VE_SIPGetUsersRegistrationData(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_sip_user_registration_s *user_registration )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the added user.
user_registration	Pointer to the structure in which the users registration details will be stored.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_AddUser**, **VE_SIPRegisterUser**, **VE_SIPAddUsersRegistrationData**, **VE_SIPRemoveUsersRegistrationData**

Description: **VE_SIPGetUsersRegistrationData** is used to obtain the SIP registration details of a user with the VeriCall Edge system. The SIP user details need to previously have been registered with the VeriCall Edge system through the **VE_SIPAddUsersRegistrationData** function.

The ve_application_id argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The users SIP registration information is stored into the structure pointed to by *user_registration*. This structure of type **t_ve_sip_user_registration_s**, is described in Appendix B.39 on Page 440.

Continued >>

Example: This example shows how to obtain a users previously added registration data:

```
#include <ve_sip_prototypes.h>

/* Updating the user's registration by expiring one of their
   contacts */
{
   t_ve_return_code_e           ret_code;
   t_ve_sip_user_registration_s ve_user_reg;

   /* Get VeriCall Edge's internal user registration data */
   ret_code = VE_SIPGetUsersRegistrationData( applicationId,
                 &ve_user_handle,
                 &ve_user_reg );
   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Assume there are two contacts, the user's home and work
      addresses:
         contacts[0] = sip:user_1@trinity.com
         contacts[1] = sip:user_1_work@work.com
      The user "user" wants to remove their work registration so
      they must tell the registration server that the work contact
      should be expired */
   strncpy( ( char* )ve_user_reg.sip_contact_info.contacts[ 1 ],
       "sip:user_1_work@work.com;expires=0",
       VE_MAX_URI_STRING_SIZE );

   /* Update VeriCall Edge's internal data with the changed user
      registration data */
   ret_code = VE_SIPUpdateUsersRegistrationData( applicationId,
               &ve_user_handle,
               &ve_user_reg );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Instruct the SIP stack to send a re-REGISTER message with the
      changed details */
   ret_code = VE_SIPUpdateRegistration( applicationId,
                                        &ve_user_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect a success or failure event */
}
```

VE_SIPUpdateUsersRegistrationData

Syntax:

```
t_ve_return_code_e  VE_SIPUpdateUsersRegistrationData(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_sip_user_registration_s *user_registration )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the added user.
user_registration	Pointer to the structure containing the users updated registration details.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_AddUser**, **VE_SIPRegisterUser**, **VE_SIPAddUsersRegistrationData**, **VE_SIPRemoveUsersRegistrationData**

Description: **VE_SIPUpdateUsersRegistrationData** is used to update the user SIP registration details of a user with the VeriCall Edge system. The SIP user details need to previously have been registered with the VeriCall Edge system through the **VE_SIPAddUsersRegistrationData** function. These user SIP registration details will also be used if the user is to be registered against a SIP registration server, performed by using the **VE_SIPRegisterUser** call.

The ve_application_id argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The users SIP registration information is held in the structure pointed to by *user_registration*. This structure of type **t_ve_sip_user_registration_s**, is described in Appendix B.39 on Page 440.

Continued >>

Example: This example shows how to update a users previously added registration data:

```
#include <ve_sip_prototypes.h>

/* Updating the user's registration by expiring one of their
   contacts */
{
   t_ve_return_code_e           ret_code;
   t_ve_sip_user_registration_s ve_user_reg;

   /* Get VeriCall Edge's internal user registration data */
   ret_code = VE_SIPGetUsersRegistrationData( applicationId,
                 &ve_user_handle,
                 &ve_user_reg );
   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Assume there are two contacts, the user's home and work
      addresses:
         contacts[0] = sip:user_1@trinity.com
         contacts[1] = sip:user_1_work@work.com
      The user "user" wants to remove their work registration so
      they must tell the registration server that the work contact
      should be expired */
   strncpy( ( char* )ve_user_reg.sip_contact_info.contacts[ 1 ],
       "sip:user_1_work@work.com;expires=0",
       VE_MAX_URI_STRING_SIZE );

   /* Update VeriCall Edge's internal data with the changed user
      registration data */
   ret_code = VE_SIPUpdateUsersRegistrationData( applicationId,
               &ve_user_handle,
               &ve_user_reg );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Instruct the SIP stack to send a re-REGISTER message with the
      changed details */
   ret_code = VE_SIPUpdateRegistration( applicationId,
                                        &ve_user_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect a success or failure event */
}
```

VE_SIPRemoveUsersRegistrationData

Syntax:

```
t_ve_return_code_e  VE_SIPRemoveUsersRegistrationData(
                        UWord32 ve_application_id,
                        t_ve_user_handle_s *ve_user_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the added user.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_RemoveUser**, **VE_SIPDeregisterUser**, **VE_SIPAddUsersRegistrationData**

Description: **VE_SIPRemoveUsersRegistrationData** is used to remove the registration details of a user previously registered with the VeriCall Edge system. This function removes the users SIP registration details from the VeriCall Edge system, it does not de-register a user from a SIP registration server, performed through the **VE_SIPDeregisterUser** function, nor does it remove the user from the VeriCall Edge system, performed through the **VE_RemoveUser** function.

The ve_application_id argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

Continued >>

Example: This example shows how to remove a users previously added registration data:

```
#include <ve_sip_prototypes.h>

/* If the user logs out of the application, the application must
   deregister the user and all f their contacts */
{
   t_ve_return_code_e           ret_code;

   /* Initiate the SIP deregistration asking the registration
      server to expire ALL the contacts listed in the
      UsersRegistrationData. NOTE: The application could use the
      VE_SIP_DEREGISTER_ALL_CONTACTS this will deregister ALL
      contacts the registration server associates with this
      user. Be careful though, because if the user has two phones,
      one a desk phone another portable and they log out of their
      desk phone with VE_SIP_DEREGISTER_ALL_CONTACTS it will cause
      their portable to deregister also! */

   ret_code = VE_SIPDeregisterUser(applicationId,
             VE_SIP_DEREGISTER_LOCAL_CONTACTS,
             &ve_user_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect a success or failure event */

   /* Wait until a success event is seen */

   /* Clear the registration data from the data store */
   ret_code = VE_SIPRemoveUsersRegistrationData(applicationId,
                                                &user_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* The application is now free for someone else to login */
}
```

VE_SIPUpdateRegistration

Syntax:

```
t_ve_return_code_e  VE_SIPUpdateRegistration (
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the added user.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_AddUser**, **VE_SIPRegisterUser**, **VE_SIPAddUsersRegistrationData**, **VE_SIPRemoveUsersRegistrationData**

Description: **VE_SIPUpdateRegistration** is used to update the user SIP registration details of a user with the remote SIP registration server. The SIP user details need to previously have been registered with the VeriCall Edge system through the **VE_SIPAddUsersRegistrationData** or **VE_SIPUpdateUsersRegistrationData** function. These user SIP registration details will also be used if the user is to be registered against a SIP registration server, performed originally by using the **VE_SIPRegisterUser** call.

The ve_application_id argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call and is use to identify the user under consideration.

Continued >>

Example: This example shows how to update a users registration details:

```
#include <ve_sip_prototypes.h>

/* Updating the user's registration by expiring one of their
   contacts */
{
   t_ve_return_code_e           ret_code;
   t_ve_sip_user_registration_s ve_user_reg;

   /* Get VeriCall Edge's internal user registration data */
   ret_code = VE_SIPGetUsersRegistrationData( applicationId,
                 &ve_user_handle,
                 &ve_user_reg );
   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Assume there are two contacts, the user's home and work
      addresses:
         contacts[0] = sip:user_1@trinity.com
         contacts[1] = sip:user_1_work@work.com
      The user "user" wants to remove their work registration so
      they must tell the registration server that the work contact
      should be expired */
   strncpy( ( char* )ve_user_reg.sip_contact_info.contacts[ 1 ],
       "sip:user_1_work@work.com;expires=0",
       VE_MAX_URI_STRING_SIZE );

   /* Update VeriCall Edge's internal data with the changed user
      registration data */
   ret_code = VE_SIPUpdateUsersRegistrationData( applicationId,
               &ve_user_handle,
               &ve_user_reg );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Instruct the SIP stack to send a re-REGISTER message with the
      changed details */
   ret_code = VE_SIPUpdateRegistration( applicationId,
                                        &ve_user_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect a success or failure event */
}
```

VE_SIPQueryRegistration

Syntax:

```
t_ve_return_code_e  VE_SIPQueryRegistration (
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the added user.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_AddUser**, **VE_SIPRegisterUser**, **VE_SIPAddUsersRegistrationData**, **VE_SIPRemoveUsersRegistrationData**

Description: **VE_SIPQueryRegistration** is used to query the user SIP registration details of a user added to the VeriCall Edge system with a remote SIP registration server. The function is useful to determine the current contact information the SIP registration server holds on the identified user

The SIP user details need to previously have been registered with the VeriCall Edge system through the **VE_SIPAddUsersRegistrationData** function. These user SIP registration details will also be used if the user is to be registered against a SIP registration server, performed by using the **VE_SIPRegisterUser** call.

The ve_application_id argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

Continued >>

Example: This example shows how to query a registration server:

```
#include <ve_sip_prototypes.h>

/* Query the registration server for what it thinks our
   registration status is */
{
   t_ve_return_code_e           ret_code;

   /* Query registration server to send the registration status */
   ret_code = VE_SIPQueryRegistration( applicationId,
                                       &ve_user_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect a success or failure event. If successful the
      application can then call VE_SIPGetUsersRegistrationData and
      get the current registration information including all the
      contacts */
}
```

VE_SIPSessionStart

Syntax:

```
t_ve_return_code_e  VE_SIPSessionStart(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_remote_user_uri_s *ve_remote_sip_uri,
                    t_ve_sip_session_handle_s *ve_sip_session_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_remote_sip_uri	Pointer to the structure containing the URI of the SIP user to be called.
ve_sip_session_handle	Pointer to a structure to store the handle identifier of the SIP session.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_SIPSessionStop**

Description: **VE_SIPSessionStart** is used to initiate a SIP session by sending an INVITE to the remote URI.

The *ve_application_id* argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The remote SIP URI for the call is contained in the structure pointed to by *ve_remote_sip_uri*. This structure of type **t_ve_remote_user_uri_s**, is described in Appendix B.51 on Page 466.

The handle to identify this session is returned by this function and stored in the structure pointed to by *ve_sip_session_handle*. This structure of type **t_ve_sip_session_handle_s**, is described in Appendix B.49 on Page 463. The handle should be used for all further VeriCall Edge API calls relating to this session.

Continued >>

Example: This example shows how to start a SIP session:

```
#include "ve_eapi.h"

/* start a new SIP session */
{
   t_ve_return_code_e          ret_code;
   t_ve_remote_user_uri_s      remote_uri;
   t_ve_sip_session_handle_s   sip_session_handle;

   /* make sure the remote_uri structure is initialized */
   memset( &remote_uri, '\0', sizeof( t_ve_remote_user_uri_s ) );

   /* Set the URI of the person we wish to call */
   strncpy( ( char* )remote_uri.uri, "sip:user_1_work@work.com",
     VE_MAX_URI_STRING_SIZE );

   /* Optionally, set a subject */
   strncpy( ( char* )remote_uri.subject, "Help on use of VeriCall
       Edge API", VE_SIP_MAX_SUBJECT_STRING_SIZE );

   /* Optionally, set that we require provisional reliable
      acknowledgement (PRACK) */
   remote_uri.require_prack = VE_SIP_REQUIRE_PRACK_YES;

   /* Place the new SIP call */
   ret_code = VE_SIPSessionStart(applicationId,
           &ve_user_handle,
           &remote_uri,
           &sip_session_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */

   /* Refer to the events VE_USER_MSG_SIP_INVITE_RESPONSE_RX for
      the destination response to the invite and
      VE_USER_MSG_SIP_SESSION_STARTED when media becomes active */
}
```

VE_SIPSessionStart_WithConfig

Syntax:

```
t_ve_return_code_e  VE_SIPSessionStart_WithConfig(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_remote_user_uri_s *ve_remote_sip_uri,
                    t_ve_stream_configuration_s *ve_stream_config,
                    t_ve_sip_session_handle_s *ve_sip_session_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_remote_sip_uri	Pointer to the structure containing the URI of the SIP user to be called.
ve_stream_config	Pointer to the structure containing the call configuration parameters.
ve_sip_session_handle	Pointer to a structure to store the handle identifier of the SIP session.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_SIPSessionStop**

Description: **VE_SIPSessionStart_WithConfig** is used to initiate a SIP session by sending an INVITE to the remote URI. The difference with **VE_SIPSessionStart** is that the call configuration parameters are provided with this function.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The remote SIP URI for the call is contained in the structure pointed to by *ve_remote_sip_uri*. This structure of type **t_ve_remote_user_uri_s**, is described in Appendix B.51 on Page 466.

The call configuration parameters to be used with this SIP session is pointed to by the *ve_stream_config* argument. This is a pointer to a structure of type **t_ve_stream_configuration_s**, as described in Appendix B.24 on Page 298.

Continued >>

The handle to identify this session is returned by this function and stored in the structure pointed to by *ve_sip_session_handle*. This structure of type **t_ve_sip_session_handle_s**, is described in Appendix B.49 on Page 463. The handle should be used for all further VeriCall Edge API calls relating to this session.

A default stream configuration is available within the frameworks attribute types. Please refer to Appendix C on Page for 659 the default stream configuration attribute type and index.

Example: This example shows how to start a SIP session with a defined configuration:

```
#include <ve_sip_prototypes.h>

/* Get default configuration, modify it & start a SIP session */
{
   t_ve_return_code_e          ret_code;
   t_ve_remote_user_uri_s      remote_uri;
   t_ve_stream_configuration_s stream_config;
   UWord32                     loop;

   /* Set the URI of the person we wish to call */
   memset( &remote_uri, '\0', sizeof( t_ve_remote_user_uri_s ) );
   strncpy( ( char* )remote_uri.uri, "sip:user_1_work@work.com",
       VE_MAX_URI_STRING_SIZE );

   /* Change the default stream configuration */
   ret_code = VE_GetSystemAttribute(applicationId,
              VE_ATTRIBUTE_TYPE_READONLY_DEFAULT_STREAM_STRUCTURE,
              VE_SINGLETON_DATA_ENTITY,
              &stream_config,
              sizeof( t_ve_stream_configuration_s ) );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Change the direction for all descriptors to Receive Only */
   for( loop = 0; loop < stream_config.num_media_descriptors;
                                                    ++loop )
   {
      stream_config.media_descriptor[ loop ].direction =
                              VE_MEDIA_STREAM_DIRECTION_RECVONLY;
   }
```

Continued >>

```
    /* Place the new SIP call where the near end is mute */
    ret_code = VE_SIPSessionStartWithConfig(applicationId,
                &ve_user_handle,
                &remote_uri,
                &stream_config,
                &sip_session_handle );

    if(ret_code != VE_RETURN_CODE_SUCCESS)
    {
       /* Handle error */
    }
}
```

VE_SIPSessionStop

Syntax:

```
t_ve_return_code_e  VE_SIPSessionStop(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_sip_session_handle_s *ve_sip_session_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_session_handle	Pointer to the structure containing the handle of the SIP session.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPSessionStop** is used to stop a specified SIP session by sending a BYE.

The *ve_application_id* argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify this session is contained in the structure pointed to by *sip_session_handle* and was either returned through a call to **VE_SIPSessionStart**/**VE_SIPSessionStart_WithConfig** or obtained from the structure associated with the VE_USER_MSG_SIP_SESSION_STARTED message obtained via **VE_GetNextEvent**. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463.

Note that either the initiator or the recipient of a call may use this function.

Continued >>

Example: This example shows how to stop a SIP session:

```
#include "ve_eapi.h"

/* Stop out active call or Cancel out outgoing call which has
   yet to be answered */
{
   t_ve_return_code_e          ret_code;

   /* Stop the call. The sip_session_handle handle should originate
      from the output of the VE_SIPSessionStart function that
      initiated the phone call. ve_user_handle should also match
      the user handle which initiated the phone call */

   ret_code = VE_SIPSessionStop(applicationId,
         &ve_user_handle,
         &sip_session_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */

   /* Refer to the event VE_USER_MSG_SIP_SESSION_STOPPED for the
      media stops */
}
```

VE_SIPSessionResponse

Syntax:

```
t_ve_return_code_e  VE_SIPSessionResponse(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_sip_session_handle_s *ve_sip_session_handle,
                    t_ve_sip_response_code_e ve_sip_response_code,
                    t_ve_sip_tx_session_response_data_s
                                    *ve_sip_tx_session_response_data )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user under consideration.
ve_sip_session_handle	Pointer to the structure containing the handle of the SIP session under consideration.
ve_sip_response_code	A standard SIP response code number.
ve_sip_tx_session_response_data	Pointer to the structure containing the extra response information that can be sent with this SIP response.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_GetNextEvent**

Description: **VE_SIPSessionResponse** is a SIP session response to the initiator of a SIP message received via **VE_GetNextEvent**. The recipient of the call may choose the appropriate response type to inform the caller whether the call will be accepted or rejected.

The ve_application_id argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify this session is contained in the structure pointed to by *ve_sip_session_handle*. This structure of type **t_ve_sip_session_handle_s**, is described in Appendix B.49 on Page 463.

The SIP response code is held in *ve_sip_response_code* and will be a standard SIP response code as defined in RFC3261. The response code enumeration is of type **t_ve_sip_response_code_e** and is described in Appendix B.181 on Page 602.

Continued >>

The *ve_sip_tx_session_response_data* argument is a pointer to the **t_ve_sip_tx_session_response_data_s** structure as described in Appendix B.44 on Page 448. This argument allows for additional content information to be included in the response message.

Example: This example shows how to generate a SIP session response:

```
#include "ve_eapi.h"

/* Respond to an incoming call using VE_SIPSessionResponse */
{
   t_ve_return_code_e                  ret_code;
   t_ve_sip_tx_session_response_data_s   tx_session_response_data

   /* Previously we have received the VE_USER_MSG_SIP_INVITE_RX
      indicating a destination phone is trying to call a local
      user. In the example sip_invite_rx is a pointer to the
      received event data */

   /* Optionally setup some response data, in this example we
      will inform the destination we require prack */

   memset(tx_session_response_data, 0 ,
          sizeof(t_ve_sip_tx_session_response_data_s));

   tx_session_response_data.require_prack =
                                  VE_SIP_REQUIRE_PRACK_YES;

   /* Respond to the incoming call and inform the destination we
      are ringing the local phone (the VE_SIP_RESPONSE_180_RINGING
      response). Later, when the user answers, we will respond
      again saying the call has started using the response
      VE_SIP_RESPONSE_200_OK. */

   ret_code = VE_SIPSessionResponse(applicationId,
              sip_invite_rx->ve_user_handle,
              sip_invite_rx->ve_sip_session_handle,
              VE_SIP_RESPONSE_180_RINGING,
              tx_session_response_data);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */
}
```

VE_SIPSessionUpdate

Syntax:

```
t_ve_return_code_e  VE_SIPSessionUpdate(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_session_handle_s *ve_sip_call_session_handle,
        t_ve_sip_session_handle_s *ve_sip_update_session_handle,
        t_ve_stream_configuration_s *ve_stream_config )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_call_session_handle	Pointer to the structure containing the handle of the current SIP session.
ve_sip_update_session_handle	Pointer to the structure containing the handle of the updated SIP session.
ve_stream_config	Pointer to the structure containing the call configuration parameters.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPSessionUpdate** is used to update an existing SIP session with the call configuration parameters provided. This function can be called if the call is active or still being negotiated and the call is updated based on the structure contents pointed to by *ve_stream_config*. This function can be used to re-negotiate the Media type and/or change the IP address used. Please note that if this function fails then the current call will not stopped.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify this session is contained in the structure pointed to by *ve_sip_call_session_handle* and was either returned through a call to **VE_SIPSessionStart/VE_SIPSessionStart_WithConfig** or obtained from the structure associated with the VE_USER_MSG_SIP_SESSION_STARTED message obtained via **VE_GetNextEvent**. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463.

Continued >>

The updated SIP session is allocated a new handle that is provides in the structure pointed to by *ve_sip_update_session_handle*, **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463. The call configuration parameters are provided by the structure pointed to by the *ve_stream_config* argument. This structure is of type **t_ve_stream_configuration_s** and is described in Appendix B.24 on Page 298.

Example: This example shows how to update a SIP session:

```
#include "ve_eapi.h"

/* Send a SIP UPDATE request to change earlier media using
   VE_SIPSessionUpdate. In the example we will update to do receive
   only */
{
   t_ve_return_code_e           ret_code;
   t_ve_stream_configuration_s  ve_stream_config;
   UWord32                      loop;

   /* Change the default stream configuration */
   ret_code = VE_GetSystemAttribute(applicationId,
              VE_ATTRIBUTE_TYPE_READONLY_DEFAULT_STREAM_STRUCTURE,
              VE_SINGLETON_DATA_ENTITY,
              &ve_stream_config,
              sizeof( t_ve_stream_configuration_s ) );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Change the direction for all descriptors to Receive Only */
   for( loop = 0; loop < ve_stream_config.num_media_descriptors;
                                                          ++loop )
   {
      ve_stream_config.media_descriptor[ loop ].direction = \
   VE_MEDIA_STREAM_DIRECTION_RECVONLY;
   }

   /* Send the SIP update. ve_sip_call_session_handle is the
      handle for the call we are trying to update.
      ve_sip_update_session_handle is returned by
      VE_SIPSessionUpdate to identify the update session separately
      from the call */

   ret_code = VE_SIPSessionUpdate(applicationId,
            &ve_user_handle,
            &ve_sip_call_session_handle,
            &ve_sip_update_session_handle,
            &ve_stream_config);
```

Continued >>

```
    if(ret_code != VE_RETURN_CODE_SUCCESS)
    {
       /* Handle error */
    }

    /* Expect a success or failure event */

    /* Refer to the events VE_USER_MSG_SIP_UPDATE_RESPONSE_RX
       for the response to the update and
       VE_USER_MSG_SIP_SESSION_UPDATED for the update to media */
}
```

VE_SIPSessionReinvite

Syntax:

```
t_ve_return_code_e  VE_SIPSessionReinvite(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_sip_session_handle_s *ve_sip_session_handle,
                    t_ve_stream_configuration_s *ve_stream_config )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_session_handle	Pointer to the structure containing the handle of the SIP session.
ve_stream_config	Pointer to the structure containing the call configuration parameters.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPSessionReinvite** allows an existing SIP session to be updated by sending a REINVITE message. The call must be active before this function can be use. This function can be used to re-negotiate the Media type and/or change the IP address used. Please note that if this function fails then the current call will not stopped.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify this session is contained in the structure pointed to by *sip_session_handle* and was either returned through a call to **VE_SIPSessionStart**/**VE_SIPSessionStart_WithConfig** or obtained from the structure associated with the VE_USER_MSG_SIP_SESSION_STARTED message obtained via **VE_GetNextEvent**. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463.

The call configuration parameters to be used with this SIP re-invite function is pointed to by the *ve_stream_config* argument. This is a pointer to a structure of type **t_ve_stream_configuration_s**, as described in Appendix B.24 on Page 298.

Continued >>

Example: This example shows how to create a SIP re-INVITE message:

```
#include <ve_sip_prototypes.h>

/* Get the stream configuration for the existing call, modify it
   and use it to update the call */
{
   t_ve_return_code_e          ret_code;
   t_ve_remote_user_uri_s      remote_uri;
   t_ve_stream_configuration_s stream_config;
   UWord32                     loop;

   /* The call is already established */

   /* Get the last stream configuration */
   ret_code = VE_GetSystemAttribute( applicationId,
            VE_ATTRIBUTE_TYPE_READONLY_LAST_START_STREAM_STRUCTURE,
            VE_SINGLETON_DATA_ENTITY,
            &stream_config,
            sizeof( t_ve_stream_configuration_s ) );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Change the direction for all descriptors to both send and
      receive */
   for( loop = 0; loop < stream_config.num_media_descriptors;
                                                        ++loop )
   {
      stream_config.media_descriptor[ loop ].direction =
                           VE_MEDIA_STREAM_DIRECTION_SENDRECV;
   }

   /* Update the existing call with the new stream configuration */
   ret_code = VE_SIPSessionReinvite(applicationId,
              &ve_user_handle,
              &sip_session_handle,
              &stream_config );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_SIPSessionHoldOn

Syntax:

```
t_ve_return_code_e  VE_SIPSessionHoldOn(
            UWord32 ve_application_id,
            t_ve_user_handle_s *ve_user_handle,
            t_ve_sip_session_handle_s *ve_sip_session_handle,
            t_ve_sip_call_hold_method_e ve_sip_call_hold_method )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_session_handle	Pointer to the structure containing the handle of the SIP session.
ve_call_hold_method	An enumerated type that defines the hold method to be used in this function call.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_SIPSessionHoldOff**

Description: **VE_SIPSessionHoldOn** is used to place the specified SIP session call on hold. The sip session call can be taken off hold by use of the **VE_SIPSessionHoldOff** function.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify this session is contained in the structure pointed to by *sip_session_handle* and was either returned through a call to **VE_SIPSessionStart**/**VE_SIPSessionStart_WithConfig** or obtained from the structure associated with the VE_USER_MSG_SIP_SESSION_STARTED message obtained via **VE_GetNextEvent**. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463.

The SIP call hold method to be used, either as per RFC2543 or RFC3264, is specified by *ve_call_hold_method* which is an enumerated type **t_ve_sip_call_hold_method_e**, described in Appendix B.190 on Page 609.

Continued >>

Example: This example shows how to place a call on hold:

```
#include <ve_sip_prototypes.h>

/* Place the current call "On Hold" */
{
   /* Assume: sip_session_handle has originated from the output
      of the VE_SIPSessionStart function that initiated the phone
      call. ve_user_handle should also match the user handle which
      initiated the phone call */
   t_ve_return_code_e          ret_code;

   /* Instruct the Call to go on hold using the method given in
      RFC2543 */
   ret_code = VE_SIPSessionHoldOn( applicationId,
            &ve_user_handle,
            &sip_session_handle,
            VE_SIP_CALL_HOLD_METHOD_RFC_2543 );
   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect an event detailing if the SIP re-INVITE was
      successful. If it failed the call doesn't get trashed */

   /* If the event stated that the hold was successful then the
      media will have been stopped */

   /* Now to go off hold */
   ret_code = VE_SIPSessionHoldOff( applicationId,
             &ve_user_handle,
             &sip_session_handle,
             VE_SIP_CALL_HOLD_METHOD_RFC_2543 );
   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect an event detailing if the SIP re-INVITE was
      successful. If it failed the call doesn't get trashed */
}
```

VE_SIPSessionHoldOff

Syntax:

```
t_ve_return_code_e  VE_SIPSessionHoldOff(
            UWord32 ve_application_id,
            t_ve_user_handle_s *ve_user_handle,
            t_ve_sip_session_handle_s *ve_sip_session_handle,
            t_ve_sip_call_hold_method_e ve_sip_call_hold_method )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_session_handle	Pointer to the structure containing the handle of the SIP session.
ve_call_hold_method	An enumerated type that defines the hold method to be used in this function call.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_SIPSessionHoldOn**

Description: **VE_SIPSessionHoldOff** is used to take the specified SIP session call off hold. The specified SIP session call must already be in a call hold mode of operation initiated by using the **VE_SIPSessionHoldOn** function.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify this session is contained in the structure pointed to by *sip_session_handle* and was either returned through a call to **VE_SIPSessionStart/VE_SIPSessionStart_WithConfig** or obtained from the structure associated with the VE_USER_MSG_SIP_SESSION_STARTED message obtained via **VE_GetNextEvent**. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463.

The SIP call hold method to be used, either as per RFC2543 or RFC3264, is specified by *ve_call_hold_method* which is an enumerated type **t_ve_sip_call_hold_method_e**, described in Appendix B.190 on Page 609.

Continued >>

Example: This example shows how to take a call off hold:

```
#include <ve_sip_prototypes.h>

/* Place the current call "On Hold" */
{
   /* Assume: sip_session_handle has originated from the output
      of the VE_SIPSessionStart function that initiated the phone
      call. ve_user_handle should also match the user handle which
      initiated the phone call */
   t_ve_return_code_e          ret_code;

   /* Instruct the Call to go on hold using the method given in
      RFC2543 */
   ret_code = VE_SIPSessionHoldOn( applicationId,
            &ve_user_handle,
            &sip_session_handle,
            VE_SIP_CALL_HOLD_METHOD_RFC_2543 );
   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect an event detailing if the SIP re-INVITE was
      successful. If it failed the call doesn't get trashed */

   /* If the event stated that the hold was successful then the
      media will have been stopped */

   /* Now to go off hold */
   ret_code = VE_SIPSessionHoldOff( applicationId,
             &ve_user_handle,
             &sip_session_handle,
             VE_SIP_CALL_HOLD_METHOD_RFC_2543 );
   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect an event detailing if the SIP re-INVITE was
      successful. If it failed the call doesn't get trashed */
}
```

VE_SIPSessionTransfer

Syntax:

```
t_ve_return_code_e  VE_SIPSessionTransfer(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_sip_session_handle_s *ve_sip_session_handle,
                    t_ve_remote_user_uri_s *ve_remote_sip_uri,
                    t_ve_remote_user_uri_s *refer_to )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_session_handle	Pointer to the structure containing the handle of the SIP session.
ve_remote_sip_uri	Pointer to the structure containing the remote SIP URI that the call is to be transferred to.
refer_to	Pointer to the structure containing the SIP URI that the SIP session is to be transferred to.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPSessionTransfer** is used to allow the user application to transfer a SIP call. Note that this is used by the user application to transfer a call when it does not have a connection with either party.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify the SIP session is contained in the structure pointed to by *sip_session_handle* and was either returned through a call to **VE_SIPSessionStart**/**VE_SIPSessionStart_WithConfig** or obtained from the structure associated with the VE_USER_MSG_SIP_SESSION_STARTED message obtained via **VE_GetNextEvent**. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463.

Continued >>

The current SIP URI of the call to be transferred is pointed to by the *ve_remote_sip_uri* argument. The SIP URI to which the call should be transferred to is pointed to by *refer_to*. Both of these arguments are pointers to the structure **t_ve_remote_user_uri_s** described in Appendix B.51 on Page 466.

It is recommended that the user application controls SIP transfer functionality with the use of the SIP hold API functions.

Example: This example shows how to transfer a call:

```
#include "ve_eapi.h"

/* Start a call transfer */
{
   t_ve_return_code_e          ret_code;
   t_ve_remote_user_uri_s     transferee;
   t_ve_remote_user_uri_s      transfer_target;

   /* Setup the transferee URI */
   memset(transfer_target, 0 , sizeof(t_ve_remote_user_uri_s));

   strncpy( ( char* )transferee.uri, "sip:transferee@work.com",
                      VE_MAX_URI_STRING_SIZE );
   /* Setup the transfer target uri */
   memset(transfer_target, 0 , sizeof(t_ve_remote_user_uri_s));
   strncpy( ( char* )transfer_target.uri,
                      "sip:transfer_target@work.com",
     VE_MAX_URI_STRING_SIZE );
   /* Transfer the transferee to a new destination.
      ve_user_handle is the user making the transfer.
      ve_sip_refer_session_handle is returned by
      VE_SIPSessionTransfer to identify this transfer session.
      transfer_target is the sip uri which will be transferred,
      this destination will be sent a SIP REFER request.
      transfer_target is the uri of the destination which the
      transferee will be transferred to.  Note, this API can be
      used when the user is not in a call to either the transferee
      or transfer target */

   ret_code = VE_SIPSessionTransfer(applicationId,
              &ve_user_handle,
              &ve_sip_refer_session_handle,
              &transferee,
              &transfer_target);
```

Continued >>

```
    if(ret_code != VE_RETURN_CODE_SUCCESS)
    {
       /* Handle error */
    }

    /* Expect a success or failure event */

    /* Refer to the events VE_USER_MSG_SIP_INVITE_RESPONSE_RX for
       the destination response to the invite and
       VE_USER_MSG_SIP_SESSION_STARTED when media becomes active */
}
```

VE_SIPSessionTransferConsult

Syntax:

```
t_ve_return_code_e  VE_SIPSessionTransferConsult(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_session_handle_s *ve_sip_refer_session_handle,
        t_ve_sip_session_handle_s *ve_sip_session_handle1,
        t_ve_sip_session_handle_s *ve_sip_session_handle2 )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_refer_session_handle	Pointer to the structure containing the handle of the transfer.
ve_sip_session_handle1	Pointer to the structure containing the SIP session handle of the call being transferred.
ve_sip_session_handle2	Pointer to the structure containing the SIP session handle to which the call is to be transferred to.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPSessionTransferConsult** is used to perform a SIP session call transfer with consultation. This allows a current SIP session call, connected to the VeriCall Edge system, to be transferred from one remote SIP URI to another remote SIP URI with consultation (attended).

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify this SIP session transfer is returned in the structure pointed to by *ve_sip_refer_session_handle*. The handles of the SIP session being transferred is contained in the structure pointed to by *ve_sip_session_handle1*, which was either returned through a call to **VE_SIPSessionStart**/**VE_SIPSessionStart_WithConfig** or obtained from the structure associated with the VE_USER_MSG_SIP_SESSION_STARTED message obtained via **VE_GetNextEvent**.

Continued >>

The handle of the SIP session to which the call is to be transferred to is pointed to by *ve_sip_session_handle2*. These arguments all point to the **t_ve_sip_session_handle_s** structure which is described in Appendix B.49 on Page 463.

It is recommended that the user application controls SIP transfer functionality with the use of the SIP hold API functions.

Example: This example shows haw to transfer with consultation:

```
#include "ve_eapi.h"

/* Start a call transfer with consultation (attended) */
{
   t_ve_return_code_e          ret_code;

   /* Transfer the first existing call to the second existing call.
      ve_user_handle is the user making the transfer.
      ve_sip_refer_session_handle is returned by
      VE_SIPSessionTransferConsult to identify this transfer
      session. sip_session_handle_transferee is the call handle to
      the call which will be transferred, this destination will be
      sent a SIP REFER request. sip_session_handle_target is the
      call handle to the call which the transferee will be
      transferred to */

   ret_code = VE_SIPSessionTransferConsult(applicationId,
                &ve_user_handle,
                &ve_sip_refer_session_handle,
                &sip_session_handle_transferee,
                &sip_session_handle_target);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */

   /* Refer to the events VE_USER_MSG_SIP_INVITE_RESPONSE_RX for
      the destination response to the invite and
      VE_USER_MSG_SIP_SESSION_STARTED when media becomes active */
}
```

VE_SIPSessionTransferDontConsult

Syntax:

```
t_ve_return_code_e  VE_SIPSessionTransferDontConsult(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_session_handle_s *ve_sip_refer_session_handle,
        t_ve_sip_session_handle_s *ve_sip_session_handle,
        t_ve_remote_user_uri_s *refer_to )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_refer_session_handle	Pointer to the structure containing the handle of the transfer.
ve_sip_session_handle	Pointer to the structure containing the SIP session handle of the call being transferred.
refer_to	Pointer to the structure containing the SIP session handle to which the call is to be transferred to.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPSessionTransferDontConsult** is used to perform a SIP session call transfer without consultation. This allows a current SIP session call, connected to the VeriCall Edge system, to be transferred from one remote SIP URI to another remote SIP URI without consultation (blind).

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify this SIP session transfer is returned in the structure pointed to by *ve_sip_refer_session_handle*. The handles of the SIP session being transferred is contained in the structure pointed to by *ve_sip_session_handle*, which was either returned through a call to **VE_SIPSessionStart**/**VE_SIPSessionStart_WithConfig** or obtained from the structure associated with the VE_USER_MSG_SIP_SESSION_STARTED message obtained via **VE_GetNextEvent**.

Continued >>

The handle of the SIP session to which the call is to be transferred to is pointed to by *refer_to*. These arguments all point to the **t_ve_sip_session_handle_s** structure which is described in Appendix B.49 on Page 463.

The remote SIP URI of the user to whom this call is to be transferred, is contained in the structure pointed to by *ve_remote_sip_uri*. This structure of type **t_ve_remote_user_uri_s**, is described in Appendix B.51 on Page 466.

It is recommended that the user application controls SIP transfer functionality with the use of the SIP hold API functions.

Example: This example shows haw to transfer without consultation:

```
#include "ve_eapi.h"

/* Start a call transfer without consultation (blind) */
{
   t_ve_return_code_e          ret_code;

   /* Transfer the first existing call to the second existing call.
      ve_user_handle is the user making the transfer.
      ve_sip_refer_session_handle is returned by
      VE_SIPSessionTransferConsult to identify this transfer
      session. sip_session_handle_transferee is the call handle to
      the call which will be transferred, this destination will be
      sent a SIP REFER request. sip_session_handle_target is the
      call handle to the call which the transferee will be
      transferred to */

   ret_code = VE_SIPSessionTransferDontConsult(applicationId,
                  &ve_user_handle,
                  &ve_sip_refer_session_handle,
                  &sip_session_handle_transferee,
                  &transfer_target);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */

   /* Refer to the events VE_USER_MSG_SIP_INVITE_RESPONSE_RX for
      the destination response to the invite and
      VE_USER_MSG_SIP_SESSION_STARTED when media becomes active */
}
```

VE_SIPSendMessage

Syntax:

```
t_ve_return_code_e  VE_SIPSendMessage(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_session_handle_s *ve_sip_call_session_handle,
        t_ve_sip_session_handle_s *ve_sip_message_session_handle,
        t_ve_remote_user_uri_s *ve_remote_sip_uri,
        t_ve_sip_content_data_s *ve_sip_content_data )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_call_session_handle	Pointer to a structure containing the handle of the call to associate the instant message with.
ve_sip_message_session_handle	Pointer to the structure containing the handle associated with the instant message being sent.
ve_remote_sip_uri	Pointer to the structure containing the current remote SIP URI.
ve_sip_content_data	Pointer to the structure containing the instant message data.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPSendMessage** is used to send an Instant Message (IM) to a remote SIP URI. The IM can be optionally associated with an existing SIP session, in which case the IM is sent within the existing dialog. Alternatively the IM can be sent outside any existing dialogs. This function also allows for split content to be sent where if the content of the IM is larger than the allowed maximum data size, then multiple calls to this function can be made to pass the IM fragments to the SIP stack. The SIP stack will only send the IM when the final fragment of the IM has been received.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

Continued >>

The handle to identify this session is contained in the structure pointed to by *sip_session_handle* and was either returned through a call to **VE_SIPSessionStart/VE_SIPSessionStart_WithConfig** or obtained from the structure associated with the VE_USER_MSG_SIP_SESSION_STARTED message obtained via **VE_GetNextEvent**. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463.

The remote SIP URI is the destination of the IM and is pointed to by the *ve_remote_sip_uri* argument, which is a pointer to the **t_ve_remote_user_uri_s** structure described in Appendix B.51 on Page 466.

The IM data to be sent is pointed to by *ve_sip_content_data*, which utilizes the **t_ve_sip_content_data_s** structure as described in Appendix B.45.2 on Page 450. This structure allows for the fragmenting of the IM data.

Example: This example shows how to send a SIP message:

```
#include <ve_sip_prototypes.h>

/* Send an instant message to another user "User_1" */
{
   /* Assume: ve_user_handle has been stored previously */
   t_ve_return_code_e          ret_code;
   t_ve_remote_user_uri_s      ve_remote_sip_uri;
   t_ve_sip_content_data_s     ve_sip_content_data;
   t_ve_sip_session_handle_s   sip_message_session_handle;
   t_ve_remote_user_uri_s      remote_uri;


   memset( &remote_uri, '\0', sizeof( t_ve_remote_user_uri_s ) );
   memset( &ve_sip_content_data, '\0', sizeof(
                                        t_ve_sip_content_data_s ) );

   /* Set the URI of the person we want to send the message to */
   strncpy( ( char* )remote_uri.uri, "sip:User_1@work.com",
                                           VE_MAX_URI_STRING_SIZE );

   /* Set the type of content to be sent */
   strncpy( ve_sip_content_data.content_type, "text",
                       VE_SIP_MAX_CONTENT_DATA_CONTENT_TYPE_LEN );

   /* Write the message to be sent */
   strncpy( ve_sip_content_data.content,
                    "Have you seen VeriCall Edge in operation?",
                           VE_SIP_MAX_CONTENT_DATA_CONTENT_LEN );

   /* Set the content length */
   ve_sip_content_data.content_length =
                         strlen( &ve_sip_content_data.content[0] );
```

Continued >>

```
    /* Send the message       */
    ret_code = VE_SIPSendMessage( applicationId,
           &ve_user_handle,
           NULL,
           &sip_message_session_handle,
           &ve_remote_sip_uri,
           &ve_sip_content_data );

    if(ret_code != VE_RETURN_CODE_SUCCESS)
    {
       /* Handle error */
    }
    /* Expect an event detailing if the SIP sending of the
       message was successful */
}
```

VE_SIPSendOptions

Syntax:

```
t_ve_return_code_e VE_SIPSendOptions(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_session_handle_s *ve_sip_call_session_handle,
        t_ve_sip_session_handle_s *ve_sip_options_session_handle,
        t_ve_remote_user_uri_s *ve_remote_sip_uri,
        t_ve_sip_content_data_s *ve_sip_content_data)
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_call_session_handle	Pointer to a structure containing the handle of the call to associate the message with. May be NULL (see description below).
ve_sip_options_session_handle	Pointer to the structure containing the handle associated with the OPTIONS message being sent.
ve_remote_sip_uri	Pointer to the structure containing the current remote SIP URI.
ve_sip_content_data	Pointer to the structure containing the message data.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes are provided in Table A.1 on Page 245.

See Also: **VE_AddUser**

Description: **VE_SIPSendOptions** is used to send a SIP OPTIONS message to the specified remote SIP destination. The SIP OPTIONS message can optionally be associated with an existing SIP session by setting *ve_sip_call_session_handle* to the handle of the associated call. This will cause the message to be sent within the existing dialog. Alternatively the message can be sent outside any existing dialogs by setting *ve_sip_call_session_handle* to NULL. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463.

The *ve_application_id* argument must be a valid registered application identifier.

The handle to identify the user is contained in a structure pointed to by *ve_user_handle*, and must be that returned from a **VE_AddUser** function call. The structure is of type **t_ve_user_handle_s** as described in Appendix B.25 on Page 299.

Continued >>

The handle to identify the OPTIONS session is contained in a structure pointed to by *ve_sip_options_session_handle*. This allows the user to uniquely identify the OPTIONS request. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463.

The destination of the message is identified by *ve_remote_sip_uri*, a pointer to a structure of type **t_ve_remote_user_uri_s** as described in Appendix B.51 on Page 466. This structure contains the remote SIP URI.

The message data to be sent is identified by *ve_sip_content_data*, a pointer to a structure of type **t_ve_sip_content_data_s** as described in Appendix B.45.2 on Page 450. This structure allows for fragmenting of the OPTIONS message data.

Example: This example shows how to send a SIP OPTIONS message:

```
#include <ve_sip_prototypes.h>

/* Send an SIP OPTIONS request to another user "User_1" */
{
   /* Assume: ve_user_handle has been stored previously */
   t_ve_return_code_e          ret_code;
   t_ve_remote_user_uri_s      ve_remote_sip_uri;
   t_ve_sip_content_data_s     ve_sip_content_data;
   t_ve_sip_session_handle_s   sip_options_session_handle;
   t_ve_remote_user_uri_s      remote_uri;

   memset( &remote_uri, '\0', sizeof( t_ve_remote_user_uri_s ) );
   memset( &ve_sip_content_data, '\0', sizeof(
                                       t_ve_sip_content_data_s ) );

   /* Set the URI of the person we want to send the message to */
   strncpy( ( char* )remote_uri.uri, "sip:User_1@work.com",
                                       VE_MAX_URI_STRING_SIZE );

   /* Set the type of content to be sent */
   strncpy( ve_sip_content_data.content_type, "text",
                       VE_SIP_MAX_CONTENT_DATA_CONTENT_TYPE_LEN );

   /* Write the message to be sent */
   strncpy( ve_sip_content_data.content,
                  "Hi, have you seen VeriCall Edge in operation?",
                            VE_SIP_MAX_CONTENT_DATA_CONTENT_LEN );

   /* Set the content length */
   ve_sip_content_data.content_length =
                       strlen( &ve_sip_content_data.content[0] );
```

Continued >>

```
    /* Send the message */
    ret_code = VE_SIPSendOptions( applicationId,
             &ve_user_handle,
             NULL,
             &sip_options_session_handle,
             &ve_remote_sip_uri,
             &ve_sip_content_data );

    if(ret_code != VE_RETURN_CODE_SUCCESS)
    {
       /* Handle error */
    }
    /* Expect an event detailing if the SIP sending of the
       message was successful */
}
```

VE_SIPSendInfo

Syntax:

```
t_ve_return_code_e  VE_SIPSendInfo(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_session_handle_s *ve_sip_call_session_handle,
        t_ve_sip_session_handle_s *ve_sip_info_session_handle,
        t_ve_remote_user_uri_s *ve_remote_sip_uri,
        t_ve_sip_content_data_s *ve_sip_content_data )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_call_session_handle	Pointer to a structure containing the handle of the call to associate the message with.
ve_sip_info_session_handle	Pointer to the structure containing the handle associated with the message being sent.
ve_remote_sip_uri	Pointer to the structure containing the current remote SIP URI.
ve_sip_content_data	Pointer to the structure containing the message data.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPSendInfo** is used to send a SIP INFO message to the specified remote SIP destination. The SIP INFO message can be optionally associated with an existing SIP session, in which case the message is sent within the existing dialog. Alternatively the message can be sent outside any existing dialogs. This function also allows for split content to be sent where if the content of the message is larger than the allowed maximum data size, then multiple calls to this function can be made to pass the IM fragments to the SIP stack. The SIP stack will only send the IM when the final fragment of the IM has been received.

The *ve_application_id* argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

Continued >>

The handle to identify this session is contained in the structure pointed to by *sip_session_handle* and was either returned through a call to **VE_SIPSessionStart/VE_SIPSessionStart_WithConfig** or obtained from the structure associated with the VE_USER_MSG_SIP_SESSION_STARTED message obtained via **VE_GetNextEvent**. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463.

The remote SIP URI is the destination of the message and is pointed to by the *ve_remote_sip_uri* argument, which is a pointer to the **t_ve_remote_user_uri_s** structure described in Appendix B.51 on Page 466.

The IM data to be sent is pointed to by *ve_sip_content_data*, which utilizes the **t_ve_sip_content_data_s** structure as described in Appendix B.45.2 on Page 450. This structure allows for the fragmenting of the IM data.

Example: This example shows how to send a SIP INFO message:

```
#include <ve_sip_prototypes.h>

/* Send an SIP INFO message to another user "User_1" */
{
   /* Assume: ve_user_handle has been stored previously */
   t_ve_return_code_e          ret_code;
   t_ve_remote_user_uri_s      ve_remote_sip_uri;
   t_ve_sip_content_data_s     ve_sip_content_data;
   t_ve_sip_session_handle_s   sip_message_session_handle;
   t_ve_remote_user_uri_s      remote_uri;

   memset( &remote_uri, '\0', sizeof( t_ve_remote_user_uri_s ) );
   memset( &ve_sip_content_data, '\0', sizeof(
                                       t_ve_sip_content_data_s ) );

   /* Set the URI of the person we want to send the message to */
   strncpy( ( char* )remote_uri.uri, "sip:User_1@work.com",
                                       VE_MAX_URI_STRING_SIZE );

   /* Set the type of content to be sent */
   strncpy( ve_sip_content_data.content_type, "text",
                        VE_SIP_MAX_CONTENT_DATA_CONTENT_TYPE_LEN );

   /* Write the message to be sent */
   strncpy( ve_sip_content_data.content,
                  "Hi, have you seen VeriCall Edge in operation?",
                              VE_SIP_MAX_CONTENT_DATA_CONTENT_LEN );

   /* Set the content length */
   ve_sip_content_data.content_length =
                        strlen( &ve_sip_content_data.content[0] );
```

Continued >>

```
    /* Send the message */
    ret_code = VE_SIPSendInfo( applicationId,
             &ve_user_handle,
             NULL,
             &sip_message_session_handle,
             &ve_remote_sip_uri,
             &ve_sip_content_data );

    if(ret_code != VE_RETURN_CODE_SUCCESS)
    {
       /* Handle error */
    }
    /* Expect an event detailing if the SIP sending of the
       message was successful */
}
```

VE_SIPSendService

Syntax:

```
t_ve_return_code_e VE_SIPSendService(
      UWord32 ve_application_id,
      t_ve_user_handle_s *ve_user_handle,
      t_ve_sip_session_handle_s *ve_sip_service_session_handle,
      t_ve_remote_user_uri_s *ve_remote_sip_uri,
      t_ve_sip_content_data_s *ve_sip_content_data)
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_service_session_handle	Pointer to a structure containing the handle of the call to associate the message with.
ve_remote_sip_uri	Pointer to the structure containing the current remote SIP URI.
ve_sip_content_data	Pointer to the structure containing the message data.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes are provided in Table A.1 on Page 245.

See Also: **VE_AddUser**

Description: **VE_SIPSendService** is used to send a SIP SERVICE message to the specified remote SIP destination.

The *ve_application_id* argument must be a valid registered application identifier.

The handle to identify the user is contained in a structure pointed to by *ve_user_handle*, and must be that returned from a **VE_AddUser** function call. The structure is of type **t_ve_user_handle_s** as described in Appendix B.25 on Page 299.

The handle to identify the SERVICE session is contained in a structure pointed to by *ve_sip_service_session_handle*. This allows the user to uniquely identify the SERVICE request. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463.

The destination of the message is identified by *ve_remote_sip_uri*, a pointer to a structure of type **t_ve_remote_user_uri_s** as described in Appendix B.51 on Page 466. This structure contains the remote SIP URI.

Continued >>

The message data to be sent is identified by *ve_sip_content_data*, a pointer to a structure of type **t_ve_sip_content_data_s** as described in Appendix B.45.2 on Page 450. This structure allows for fragmenting of the SERVICE message data.

Example: This example shows how to send a SIP SERVICE message:

```
#include <ve_sip_prototypes.h>

/* Send an SIP SERVICE request to another user "User_1" */
{
   /* Assume: ve_user_handle has been stored previously */
   t_ve_return_code_e          ret_code;
   t_ve_remote_user_uri_s      ve_remote_sip_uri;
   t_ve_sip_content_data_s     ve_sip_content_data;
   t_ve_sip_session_handle_s   sip_service_session_handle;
   t_ve_remote_user_uri_s      remote_uri;

   memset( &remote_uri, '\0', sizeof( t_ve_remote_user_uri_s ) );
   memset( &ve_sip_content_data, '\0', sizeof(
                                       t_ve_sip_content_data_s ) );

   /* Set the URI of the person we want to send the message to */
   strncpy( ( char* )remote_uri.uri, "sip:User_1@work.com",
                                       VE_MAX_URI_STRING_SIZE );

   /* Set the type of content to be sent */
   strncpy( ve_sip_content_data.content_type, "text",
                        VE_SIP_MAX_CONTENT_DATA_CONTENT_TYPE_LEN );

   /* Write the message to be sent */
   strncpy( ve_sip_content_data.content,
                  "Hi, have you seen VeriCall Edge in operation?",
                              VE_SIP_MAX_CONTENT_DATA_CONTENT_LEN );

   /* Set the content length */
   ve_sip_content_data.content_length =
                        strlen( &ve_sip_content_data.content[0] );

   /* Send the message */
   ret_code = VE_SIPSendService( applicationId,
            &ve_user_handle,
            &sip_service_session_handle,
            &ve_remote_sip_uri,
            &ve_sip_content_data );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect an event detailing if the SIP sending of the
      message was successful */
}
```

VE_SIPAddUsersPresenceData

Syntax:

```
t_ve_return_code_e  VE_SIPAddUsersPresenceData(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_presence_data_s *ve_user_presence_data )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_user_presence_data	Pointer to the structure containing the users presence data.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPAddUsersPresenceData** is used to add the users presence information. This information will be relayed on to the destinations that have subscribed to the local users presence.

The *ve_application_id* argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The user presence information to be added is contained in the structure pointed to by *ve_user_presence_data*, which is a structure of type **t_ve_sip_presence_data_s** as described in Appendix B.46 on Page 452.

Continued >>

Example: This example shows how to add a user presence data:

```
#include "ve_sip_prototypes.h"

/* Add a users presence details to VeriCall Edge */
{
   t_ve_return_code_e       ret_code;
   t_ve_sip_presence_data_s user_presence_data

   user_presence_data.type = VE_SIP_PRESENCE_TYPE_XML_PIDF;
   strncpy(user_presence_data.data.xml_pidf.entity,
                   "joe@blogs.com", VE_MAX_URI_STRING_SIZE);
   user_presence_data.data.xml_pidf.num_tuples = 1;
   strncpy(user_presence_data.data.xml_pidf.tuple[0].id , "id1",
                                                  VE_SIP_MAX_ID_LEN);
   user_presence_data.data.xml_pidf.tuple[0].status.basic =
                        VE_SIP_PRESENCE_XML_PIDF_STATUS_BASIC_OPEN;
   user_presence_data.data.xml_pidf.tuple[0].contact.priority =
                                                              500;
   strncpy(user_presence_data.data.xml_pidf.tuple[0].contact.value,
                     "SIP:joe@blogs.com", VE_MAX_URI_STRING_SIZE);
   strncpy(user_presence_data.data.xml_pidf.tuple[0].note.language,
                                   "EN", VE_SIP_MAX_LANGUAGE_LEN);
   strncpy(user_presence_data.data.xml_pidf.tuple[0].note.value ,
                                    "Hello!", VE_SIP_MAX_NOTE_LEN);
   user_presence_data.data.xml_pidf.num_notes = 1;
   strncpy(user_presence_data.data.xml_pidf.note[0].language ,
                                    "EN", VE_SIP_MAX_LANGUAGE_LEN);
   strncpy(user_presence_data.data.xml_pidf.note[0].value ,
                                 "Note Hello!", VE_SIP_MAX_NOTE_LEN);
   /* Note - don't need to fill in timestamp as Edge will do that
      automatically */

   ret_code = VE_SIPAddUsersPresenceData( application_id,
              &user_handle,
              &user_presence_data);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_SIPGetUsersPresenceData

Syntax:

```
t_ve_return_code_e  VE_SIPGetUsersPresenceData(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_presence_data_s *ve_user_presence_data )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_user_presence_data	Pointer to the structure containing the users presence data.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPGetUsersPresenceData** used to retrieve the users presence information.

The *ve_application_id* argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The retrieved user presence information is stored in the structure pointed to by *ve_user_presence_data*, which is a structure of type **t_ve_sip_presence_data_s** as described in Appendix B.46 on Page 452.

Continued >>

Example: This example shows how to obtain a users presence data:

```
#include "ve_sip_prototypes.h"

/* Get a users presence details from VeriCall Edge */
{
   t_ve_return_code_e ret_code;
   t_ve_sip_presence_data_s user_presence_data

      ret_code = VE_SIPGetUsersPresenceData( application_id,
                 &user_handle,
                 &user_presence_data);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_SIPUpdateUsersPresenceData

Syntax:

```
t_ve_return_code_e  VE_SIPUpdateUsersPresenceData(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_presence_data_s *ve_user_presence_data )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_user_presence_data	Pointer to the structure containing the users presence data.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPUpdateUsersPresenceData** is used to update the users presence information. This updated information will be relayed on to the destinations that have subscribed to the local users presence.

The *ve_application_id* argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The user presence information to be updated is contained in the structure pointed to by *ve_user_presence_data*, which is a structure of type **t_ve_sip_presence_data_s** as described in Appendix B.46 on Page 452.

Continued >>

Example: This example shows how to update a users presence data:

```
#include "ve_sip_prototypes.h"

/* Update a users presence details in VeriCall Edge */
{
   t_ve_return_code_e ret_code;
   t_ve_sip_presence_data_s user_presence_data

      /* Get the current data */
      ret_code = VE_SIPGetUsersPresenceData( application_id,
                &user_handle,
                &user_presence_data);

   /* Modify the details */
   user_presence_data.data.xml_pidf.tuple[0].status.basic =
                 VE_SIP_PRESENCE_XML_PIDF_STATUS_BASIC_CLOSED;

   /* Update the details */
   ret_code = VE_SIPUpdateUsersPresenceData(application_id,
                &user_handle,
                &user_presence_data);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_SIPRemoveUsersPresenceData

Syntax:
```
t_ve_return_code_e  VE_SIPRemoveUsersPresenceData(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPRemoveUsersPresenceData** is used to remove the users presence data.

The *ve_application_id* argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

Example: This example shows how to remove a users presence data:

```
#include "ve_sip_prototypes.h"

/* Remove a users presence details from VeriCall Edge */
{
   t_ve_return_code_e ret_code;

   ret_code = VE_SIPRemoveUsersPresenceData(application_id,
                &user_handle);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_SIPStartPresenceSubscription

Syntax:

```
t_ve_return_code_e  VE_SIPStartPresenceSubscription(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_session_handle_s *ve_sip_session_handle,
        t_ve_remote_user_uri_s *ve_remote_sip_uri )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_session_handle	Pointer to the structure containing the handle of the SIP session.
ve_remote_sip_uri	Pointer to the structure containing the remote URI of the users presence information.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPStartPresenceSubscription** allows a local user to SUBSCRIBE to a remote users presence information. When the presence information of the remote user is received by the VeriCall Edge system, this will be conveyed to the user application by the VE_USER_MSG_SIP_PRESENCE_RX event.

The ve_application_id argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify the Presence subscription session is returned by this function and stored in the structure pointed to by *ve_sip_session_handle*. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463. The returned handle should be used fir all further API call relating to this Message Waiting subscription session.

The remote SIP URI of the user whose presence data is to be subscribed to, is contained in the structure pointed to by *ve_remote_sip_uri*. This structure of type **t_ve_remote_user_uri_s**, is described in Appendix B.51 on Page 466.

Continued >>

Example: This example shows how to SUBSCRIBE to a users presence:

```
#include "ve_sip_prototypes.h"

/* Subscribe to receive notifications about someone else's
   presence details */
{
   t_ve_return_code_e        ret_code;
   t_ve_sip_session_handle_s presence_session_handle;
   t_ve_remote_user_uri_s    remote_sip_uri;

   strncpy(remote_sip_uri.uri, "SIP:joe@harry.net",
                                       VE_MAX_URI_STRING_SIZE);

   ret_code = VE_SIPStartPresenceSubscription( application_id,
                  &user_handle,
                  &presence_session_handle,
                  &remote_sip_uri);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_SIPStopPresenceSubscription

Syntax:

```
t_ve_return_code_e  VE_SIPStopPresenceSubscription(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_session_handle_s *ve_sip_session_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_session_handle	Pointer to the structure containing the handle of the SIP session.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPStopPresenceSubscription** allows a local user to un-SUBSCRIBE from the specified remote users presence information. Use of this function will result in the local user no longer receiving presence information about the remote user.

The *ve_application_id* argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify this session is contained in the structure pointed to by *ve_sip_session_handle* and was returned through a call to **VE_SIPStartPresenceSubscription**. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463.

Continued >>

Example: This example shows how to un-SUBSCRIBE to a users presence:

```
#include "ve_sip_prototypes.h"

/* Stop a presence subscription */
{
   t_ve_return_code_e ret_code;

   /* Presence_session_handle generated by
      VE_SIPStartPresenceSubscription */
   ret_code = VE_SIPStopPresenceSubscription( application_id,
                 &user_handle,
                 &presence_session_handle);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_SIPStartMessageWaitingSubscription

Syntax:

```
t_ve_return_code_e  VE_SIPStartMessageWaitingSubscription(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_session_handle_s *ve_sip_session_handle,
        t_ve_remote_user_uri_s *ve_remote_sip_uri,
        t_ve_sip_mw_start_flag_e ve_sip_mw_start_flag )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_session_handle	Pointer to the structure containing the handle of the SIP session.
ve_remote_sip_uri	Pointer to the structure containing the remote URI destination.
ve_sip_mw_start_flag	An enumerated type used as a directive to start the Message Waiting subscription.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPStartMessageWaitingSubscription** is used to start a SIP Message Waiting subscription.

The *ve_application_id* argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify the Message Waiting subscription session is returned by this function and stored in the structure pointed to by *ve_sip_session_handle*. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463. The returned handle should be used for all further API call relating to this Message Waiting subscription session.

The remote SIP URI destination for the Message Waiting subscription, is contained in the structure pointed to by *ve_remote_sip_uri*. This structure of type **t_ve_remote_user_uri_s**, is described in Appendix B.51 on Page 466.

Continued >>

A flag to indicate the method of Message Waiting subscription is passed to the function through the *ve_sip_mw_start_flag* enumerated argument. The enumeration, **t_ve_sip_mw_start_flag_e**, is given in Appendix B.196 on Page 616.

Example: This example shows how to SUBSCRBE to message waiting:

```
#include <ve_sip_prototypes.h>

/* Subscribe to a Message Waiting server */
{
   /* Assume sip_mw_session_handle is stored away somewhere in the
      application */

   t_ve_return_code_e          ret_code;
   t_ve_remote_user_uri_s      remote_uri;

   /* The SIP stack may be registered with another user and
      password so override this by setting the URI of the account
      and mailbox server (with the user's work account password) */
   memset( &remote_uri, '\0', sizeof( t_ve_remote_user_uri_s ) );
   strncpy( ( char* )remote_uri.uri,"sip:user_1_work@work.com",
                                          VE_MAX_URI_STRING_SIZE );
   strncpy( ( char* )remote_uri.password,  "Password",
                                          VE_MAX_PASSWORD_SIZE );

   /* Subscribe to "user_1_work" mailbox which is served by
      work.com. Also instruct the SIP stack to follow RFC3842, that
      is the correct method of message-summary SUBSCRIPTION rather
      than allowing unsolicited NOTIFY message-summary messages */

   ret_code = VE_SIPStartMessageWaitingSubscription( applicationId,
                  &ve_user_handle,
                  &sip_mw_session_handle,
                  &remote_uri,
                  VE_SIP_MESSAGES_WAITING_START_RFC3842 );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* The application will now receive NOTIFY message-summary
      events when any change is made to "user_1_work" mailbox
      make sure sip_mw_session_handle is kept for future operations
      on this Message Waiting subscription */
}
```

VE_SIPStopMessageWaitingSubscription

Syntax:

```
t_ve_return_code_e  VE_SIPStopMessageWaitingSubscription(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_session_handle_s *ve_sip_session_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_session_handle	Pointer to the structure containing the handle of the SIP session.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPStopMessageWaitingSubscription** is used to stop a SIP Message Waiting subscription.

The *ve_application_id* argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify this session is contained in the structure pointed to by *ve_sip_session_handle* and was returned through a call to **VE_SIPStartMessageWaitingSubscription**. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463.

Continued >>

Example: This example shows how to un-SUBSCRBE to message waiting:

```
#include <ve_sip_prototypes.h>

/* Un-subscribe from a Message Waiting server */
{
   /* Assume sip_mw_session_handle is stored away somewhere in
      the application */

   t_ve_return_code_e          ret_code;

   /* Un-subscribe from an existing mw subscription to a mailbox
      server.
      * Note: the sip_mw_session_handle is different from that of
      * sip_session_handle used in other SIP functions */

   ret_code = VE_SIPStopMessageWaitingSubscription(applicationId,
                  &ve_user_handle,
                  &sip_mw_session_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_SIPInterrogateMessageWaiting

Syntax:

```
t_ve_return_code_e  VE_SIPInterrogateMessageWaiting(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_session_handle_s *ve_sip_session_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_sip_session_handle	Pointer to the structure containing the handle of the SIP session.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_SIPInterrogateMessageWaiting** is used to interrogate a users SIP Message Waiting subscription. After this function has been called, the user application can expect a NOTIFY message containing the current mailbox status for the requested subscription

The *ve_application_id* argument must be a valid registered application identifier.

The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify this session is contained in the structure pointed to by *ve_sip_session_handle* and was returned through a call to **VE_SIPStartMessageWaitingSubscription**. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463.

Continued >>

Example: This example shows how to query a message waiting server:

```
#include <ve_sip_prototypes.h>

/* Interrogate a subscription to a Message Waiting server */
{
   /* Assume sip_mw_session_handle is stored away somewhere in
      the application */

   t_ve_return_code_e          ret_code;

   /* Interrogate an existing mw subscription to a mailbox server
      * Note: the sip_mw_session_handle is different from that of
      * sip_session_handle used in other SIP functions */
   ret_code = VE_SIPInterrogateMessageWaiting( applicationId,
                  &ve_user_handle,
                  &sip_mw_session_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Now expect a NOTIFY with the current subscription's
      mailbox status */
}
```

VE_SIPStartDialogSubscription

Syntax:

```
t_ve_return_code_e VE_SIPStartDialogSubscription(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_session_handle_s *ve_sip_session_handle,
        t_ve_remote_user_uri_s *ve_remote_sip_uri )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to a structure containing the handle of the user.
ve_sip_session_handle	Pointer to a structure. A handle to the established dialog session will be returned to this structure.
ve_remote_sip_uri	Pointer to a structure containing the remote SIP URI of the user and host to which the subscription is to be sent.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes are provided in Table A.1 on Page 245.

See Also: VE_SIPStopDialogSubscription, VE_SIPInterrogateDialogSubscription

Description: **VE_SIPStartDialogSubscription** sends a message to the destination requesting a SIP dialog subscription so that notification of SIP INVITE established dialog information can be received.

The *ve_application_id* argument must be a valid registered application identifier.

The handle to identify the user is contained in a structure pointed to by *ve_user_handle*, and must be that returned from a **VE_AddUser** function call. The structure is of type **t_ve_user_handle_s** as described in Appendix B.25 on Page 299.

The handle to the dialog subscription is identified by *ve_sip_session_handle*, a pointer to a structure of type **t_ve_sip_session_handle_s,** as described in Appendix B.49 on Page 463.

The destination of the message is identified by *ve_remote_sip_uri*, a pointer to a structure of type **t_ve_remote_user_uri_s** as described in Appendix B.51 on Page 466. This structure contains the remote SIP URI.

Continued >>

Example: This example shows how to SUBSCRIBE to dialog information:

```
#include <ve_sip_prototypes.h>

/* Subscribe to a user's dialog on another host */
{

   t_ve_return_code_e          ret_code;
   t_ve_sip_session_handle_s   ve_sip_session_handle;
   t_ve_remote_user_uri_s      remote_uri;

   /* Set the uri of the user to establish the dialog subscription
      with */
   memset( &remote_uri, '\0', sizeof( t_ve_remote_user_uri_s ) );
   strncpy( ( char* )remote_uri.uri,"sip:user_2_work@work.com",
                                          VE_MAX_URI_STRING_SIZE );
   strncpy( ( char* )remote_uri.password,  "Password2",
                                          VE_MAX_PASSWORD_SIZE );

   /* Subscribe to "user_2_work"'s dialog information.
    * Note: the sip_dialog_session_handle is different from that of
    * sip_session_handle used in other SIP functions */

   ret_code = VE_SIPStartDialogSubscription( applicationId,
                  &ve_user_handle,
                  &sip_dialog_session_handle,
                  &remote_uri );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /*  The application will now receive NOTIFY events when any
      change is made to "user_1_work"'s INVITE established
      dialogs, such as when a new SIP call is made or when a call
      ends. Make sure sip_dialog_session_handle is kept for future
      operations on this Dialog subscription.*/
}
```

VE_SIPStopDialogSubscription

Syntax:

```
t_ve_return_code_e VE_SIPStopDialogSubscription(
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_session_handle_s *ve_sip_session_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to a structure containing the handle of the user.
ve_sip_session_handle	Pointer to a structure containing a handle to the dialog session.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes are provided in Table A.1 on Page 245.

See Also: **VE_SIPStartDialogSubscription**, **VE_SIPInterrogateDialogSubscription**

Description: **VE_SIPStopDialogSubscription** stops a SIP dialog subscription. A SIP dialog subscription is started by calling function **VE_SIPStartDialogSubscription**.

The *ve_application_id* argument must be a valid registered application identifier.

The handle to identify the user is contained in a structure pointed to by *ve_user_handle*, and must be that returned from a **VE_AddUser** function call. The structure is of type **t_ve_user_handle_s** as described in Appendix B.25 on Page 299.

The handle to identify this session is contained in the structure pointed to by *ve_sip_call_session_handle* and was returned through a call to **VE_SIPStartDialogSubscription**. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463.

Continued >>

Example: This example shows how to stop a SIP dialog subscription:

```
#include <ve_sip_prototypes.h>

/* Stop the dialog subscription */
{
   /* Assume sip_dialog_session_handle was returned
      by function VE_SIPStartDialogSubscription() */

   t_ve_return_code_e          ret_code;

   /* Stop an existing dialog subscription.
    * Note: the sip_dialog_session_handle is different from that of
    * sip_session_handle used in other SIP functions */

   ret_code = VE_SIPStopDialogSubscription( applicationId,
                  &ve_user_handle,
                  &sip_dialog_session_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_SIPInterrogateDialogSubscription

Syntax:

```
t_ve_return_code_e VE_SIPInterrogateDialogSubscription (
        UWord32 ve_application_id,
        t_ve_user_handle_s *ve_user_handle,
        t_ve_sip_session_handle_s *ve_sip_session_handle,
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to a structure containing the handle of the user.
ve_sip_session_handle	Pointer to a structure containing a handle to the dialog session.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes are provided in Table A.1 on Page 245.

See Also: **VE_SIPStartDialogSubscription**, **VE_SIPStopDialogSubscription**

Description: **VE_SIPInterrogateDialogSubscription** interogates an existing dialog subscription that was started by a call to function **VE_SIPStartDialogSubscription**. This function call will result in the user (that the subscription is with) issuing a NOTIFY message containing the status of its current INVITE established SIP dialogs.

The *ve_application_id* argument must be a valid registered application identifier.

The handle to identify the user is contained in a structure pointed to by *ve_user_handle*, and must be that returned from a **VE_AddUser** function call. The structure is of type **t_ve_user_handle_s** as described in Appendix B.25 on Page 299.

The handle to identify this session is contained in the structure pointed to by *ve_sip_call_session_handle* and was returned through a call to **VE_SIPStartDialogSubscription**. The **t_ve_sip_session_handle_s** structure is described in Appendix B.49 on Page 463.

Continued >>

Example: This example shows how to query a dialog subscription:

```
#include <ve_sip_prototypes.h>

/* Interrogate a dialog subscription */
{
   /* Assume sip_dialog_session_handle was returned
      by function VE_SIPStartDialogSubscription() */

   t_ve_return_code_e          ret_code;

   /* Interrogate an existing dialog subscription.
    * Note: the sip_dialog_session_handle is different from that of
    * sip_session_handle used in other SIP functions */
   ret_code = VE_SIPInterrogateDialogSubscription( applicationId,
                  &ve_user_handle,
                  &sip_dialog_session_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Now expect a NOTIFY from the user that the subscription is
    * with.  The NOTIFY will contain the status of all the
    * user's current INVITE established dialogs. */
}
```

Chapter 8

VeriCall Edge™ H.323 API

This chapter describes the VeriCall Edge H.323 API functions. The VeriCall Edge H.323 API is used for call handling.

8.1 VeriCall Edge H.323 API Function Descriptions

For each function detailed in this section, the following are described (as appropriate):

- **Syntax** Command syntax.
- **Arguments** Description of the arguments.
- **Returns** Return codes from the function.
- **See Also** Other functions that perform related tasks.
- **Description** What the function can be used for.
- **Example** Short code sections showing how to use the function.

8.2 VeriCall Edge H.323 API Functions

The VeriCall Edge H.323 API functions are described on the following pages:

Function	Page
VE_H323RegisterUser	183
VE_H323DeregisterUser	185
VE_H323AddUsersRegistrationData	187
VE_H323UpdateUsersRegistrationData	189
VE_H323RemoveUsersRegistrationData	191
VE_H323GetUsersRegistrationData	193
VE_H323StartCall	195
VE_H323StopCall	197
VE_H323UpdateCall	199
VE_H323CallResponse	201
VE_H323SendUserInput	203
VE_H323VideoFastUpdate	205
VE_H323VideoFreeze	207
VE_H323CallTransferConsult	209
VE_H323CallTransferDontConsult	211
VE_H323CallForward	213
VE_H323CallForwardActivate	215
VE_H323CallForwardDeactivate	217
VE_H323CallHoldOn	219
VE_H323CallHoldOff	221
VE_H323MWIInterrogate	223

VE_H323RegisterUser

Syntax:

```
t_ve_return_code_e  VE_H323RegisterUser(
                        UWord32 ve_application_id,
                        t_ve_user_handle_s *ve_user_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user that has been previously added to the VeriCall Edge system.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_AddUser**, **VE_RemoveUser**, **VE_H323DeregisterUser**, **VE_H323AddUsersRegistrationData**

Description: **VE_H323RegisterUser** is used to register a user with a H.323 gatekeeper. Before this function is used, a user must have been added to the VeriCall Edge system, through the **VE_AddUser** function, and a call to **VE_H323AddUsersRegistrationData** must have been made, which contains the users H.323 registration details.

The gatekeeper against which the user is to be registered, is identified within the **t_ve_h323_ras_configuration_s** structure, as defined in Appendix B.58.2 on Page 477. This structure is part of the VeriCall Edge VE_ATTRIBUTE_TYPE_DEFAULT_H323_CONFIGURATION attribute, refer to Table C.1 on Page 659 for this attributes details.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

Continued >>

Note: In the context of H.323, VeriCall Edge users are equivalent to H.323 aliases.

Continued >>

Example: This example shows how to register a H.323 user:

```
#include <ve_h323_prototypes.h>

/* Start a new H.323 registration */
{
   t_ve_return_code_e  ret_code;
   t_ve_h323_user_registration_s user_reg;


   strncpy((char*)user_reg.h323_user.h323_user_address.address,
                        "TEL:",VE_H323_MAX_ADDRESS_LEN);
   strncat((char*)user_reg.h323_user.h323_user_address.address,
                        "5351",VE_H323_MAX_ADDRESS_LEN);

   /* set the name to be the same as the users phone number */
   strncpy((char*)user_reg.h323_user.h323_name_id.name,
               "hello, my name is 5351", VE_H323_MAX_NAME_ID_LEN);
   strncat((char*)(char*)user_reg.h323_user.h323_name_id.name,
               " from IP: ", VE_H323_MAX_NAME_ID_LEN);
   strncat((char*)(char*)user_reg.h323_user.h323_name_id.name,
               "192.168.1.101", VE_H323_MAX_NAME_ID_LEN);
   user_reg.h323_user.h323_name_id.char_set =
               VE_H323_NAME_ID_CHAR_SET_ISO8859_1;
   user_reg.h323_user.h323_name_id.restriction =
            VE_H323_NAME_ID_RESTRICTION_NAME_PRESENTATION_ALLOWED;
   /* Contact details are optional */

   /* Add the registration details of the H.323 user */
   ret_code = VE_H323AddUsersRegistrationData(ve_application_id,
                                              &user_handle,
                                              &user_reg );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Register the user to a H.323 gatekeeper */
   /* Assume auto-discovery is enabled for gatekeeper address */
   ret_code = VE_H323RegisterUser( ve_application_id,
                                   &user_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */
}
```

VE_H323DeregisterUser

Syntax:

```
t_ve_return_code_e  VE_H323DeregisterUser(
                        UWord32 ve_application_id,
                        t_ve_user_handle_s *ve_user_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to structure containing the handle of the user, previously added to the VeriCall Edge system, that is to be removed.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_AddUser**, **VE_RemoveUser**, **VE_H323RegisterUser**, **VE_H323RemoveUsersRegistrationData**

Description: **VE_H323DeregisterUser** is used to de-register a user from a H.323 Gatekeeper.

This function only de-registers the user from the H.323 gatekeeper, the user is still registered with the VeriCall Edge system.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

Example: This example shows how to de-register a H.323 user:

```
#include <ve_h323_prototypes.h>

/* If the user logs out of the application, the application must
   deregister the user */
{
   t_ve_return_code_e ret_code;

   /* Initiate the H.323 deregistration */
   ret_code = VE_H323DeregisterUser(applicationId,
              &user_handle);
```

Continued >>

```
    if(ret_code != VE_RETURN_CODE_SUCCESS)
    {
       /* Handle error */
    }
    /* Expect a success or failure event */

    /* Wait until a success event is seen */
    /* Clear the registration data from the data store */
    ret_code = VE_H323RemoveUsersRegistrationData( applicationId,
                &user_handle);

    if(ret_code != VE_RETURN_CODE_SUCCESS)
    {
       /* Handle error */
    }

    /* The application is now free for someone else to login */
}
```

VE_H323AddUsersRegistrationData

Syntax:

```
t_ve_return_code_e  VE_H323AddUsersRegistrationData(
          UWord32 ve_application_id,
          t_ve_user_handle_s *ve_user_handle,
          t_ve_h323_user_registration_s *user_registration )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the added user.
user_registration	Pointer to the structure containing the users registration details.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_AddUser**, **VE_H323RegisterUser**, **VE_H323UpdateUsersRegistrationData**, **VE_H323RemoveUsersRegistrationData**

Description: **VE_H323AddUsersRegistrationData** is used to add user H.323 registration details required by the VeriCall Edge system. The user must have been previously added to the VeriCall Edge system through the **VE_AddUser** function. These user H.323 registration details will also be used if the user is to be registered against a H.323 gatekeeper, performed by using the **VE_H323RegisterUser** call.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The users H.323 registration information is held in the structure pointed to by *user_registration*. This structure is of type **t_ve_h323_user_registration_s** and is described in Appendix B.52 on Page 467.

Continued >>

Example: This example shows how to add a H.323 users registration data:

```
#include <ve_h323_prototypes.h>

/* Start a new H.323 registration */
{
   t_ve_return_code_e  ret_code;
   t_ve_h323_user_registration_s user_reg;


   strncpy((char*)user_reg.h323_user.h323_user_address.address,
                        "TEL:",VE_H323_MAX_ADDRESS_LEN);
   strncat((char*)user_reg.h323_user.h323_user_address.address,
                        "5351",VE_H323_MAX_ADDRESS_LEN);

   /* set the name to be the same as the users phone number */
   strncpy((char*)user_reg.h323_user.h323_name_id.name,
               "hello, my name is 5351", VE_H323_MAX_NAME_ID_LEN);
   strncat((char*)(char*)user_reg.h323_user.h323_name_id.name,
               " from IP: ", VE_H323_MAX_NAME_ID_LEN);
   strncat((char*)(char*)user_reg.h323_user.h323_name_id.name,
               "192.168.1.101", VE_H323_MAX_NAME_ID_LEN);
   user_reg.h323_user.h323_name_id.char_set =
               VE_H323_NAME_ID_CHAR_SET_ISO8859_1;
   user_reg.h323_user.h323_name_id.restriction =
            VE_H323_NAME_ID_RESTRICTION_NAME_PRESENTATION_ALLOWED;
   /* Contact details are optional */

   /* Add the registration details of the H.323 user */
   ret_code = VE_H323AddUsersRegistrationData(ve_application_id,
                                              &user_handle,
                                              &user_reg );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Register the user to a H.323 gatekeeper */
   /* Assume auto-discovery is enabled for gatekeeper address */
   ret_code = VE_H323RegisterUser( ve_application_id,
                                   &user_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */
}
```

VE_H323UpdateUsersRegistrationData

Syntax:

```
t_ve_return_code_e  VE_H323UpdateUsersRegistrationData(
                UWord32 ve_application_id,
                t_ve_user_handle_s *ve_user_handle,
                t_ve_h323_user_registration_s *user_registration )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the added user.
user_registration	Pointer to the structure containing the users updated registration details.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_AddUser**, **VE_H323RegisterUser**, **VE_H323AddUsersRegistrationData**, **VE_H323RemoveUsersRegistrationData**

Description: **VE_H323UpdateUsersRegistrationData** is used to update the user H.323 registration details of a user with the VeriCall Edge system. The H.323 user details need to previously have been registered with the VeriCall Edge system through the **VE_H323AddUsersRegistrationData** function. These user H.323 registration details will also be used if the user is to be registered against a H.323 gatekeeper, performed by using the **VE_H323RegisterUser** call.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The users H.323 registration information is held in the structure pointed to by *user_registration*. This structure is of type **t_ve_h323_user_registration_s** and is described in Appendix B.52 on Page 467.

Continued >>

Example: This example shows how to update a users registration data:

```
#include <ve_h323_prototypes.h>

/* Update the users registration data */
{
   t_ve_return_code_e            ret_code;
   t_ve_h323_user_registration_s ve_user_reg;

   /* Get VeriCall Edge's internal user registration data */
   ret_code = VE_H323GetUsersRegistrationData( applicationId,
                 &ve_user_handle,
                 &ve_user_reg );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Alter registration data */
   strncat((char*)(char*)user_reg.h323_user.h323_name_id.name,
          "192.168.1.102", VE_H323_MAX_NAME_ID_LEN);

   /* Update VeriCall Edge's internal data with the changed user
      registration data */
   ret_code = VE_H323UpdateUsersRegistrationData( applicationId,
               &ve_user_handle,
               &ve_user_reg );
   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_H323RemoveUsersRegistrationData

Syntax:
```
t_ve_return_code_e  VE_H323RemoveUsersRegistrationData(
                        UWord32 ve_application_id,
                        t_ve_user_handle_s *ve_user_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the added user.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_RemoveUser**, **VE_H323DeregisterUser**, **VE_H323AddUsersRegistrationData**, **VE_H323UpdateUsersRegistrationData**

Description: **VE_H323RemoveUsersRegistrationData** is used to remove the registration details of a user previously registered with the VeriCall Edge system. This function removes the users H.323 registration details from the VeriCall Edge system, it does not de-register a user from a H.323 gatekeeper, performed through the **VE_H323DeregisterUser** function, nor does it remove the user from the VeriCall Edge system, performed through the **VE_RemoveUser** function.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

Continued >>

Example: This example shows how to remove a H.323 users registration data:

```
#include <ve_h323_prototypes.h>

/* If the user logs out of the application, the application must
   deregister the user */
{
   t_ve_return_code_e ret_code;

   /* Initiate the H.323 deregistration */
   ret_code = VE_H323DeregisterUser(applicationId,
              &user_handle);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect a success or failure event */

   /* Wait until a success event is seen */
   /* Clear the registration data from the data store */
   ret_code = VE_H323RemoveUsersRegistrationData( applicationId,
                &user_handle);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* The application is now free for someone else to login */
}
```

VE_H323GetUsersRegistrationData

Syntax:

```
t_ve_return_code_e  VE_H323GetUsersRegistrationData(
          UWord32 ve_application_id,
          t_ve_user_handle_s *ve_user_handle,
          t_ve_h323_user_registration_s *user_registration )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the added user.
user_registration	Pointer to the structure to contain the users registration details.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_H323AddUsersRegistrationData**, **VE_H323UpdateUsersRegistrationData**, **VE_H323RemoveUsersRegistrationData**

Description: **VE_H323GetUsersRegistrationData** is used to obtain the registration details of a user added and registered with the VeriCall Edge system.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The registration information returned by calling this function is stored in the t_ve_h323_user_registration_s structure pointed to by *user_registration*, this structure is described in Appendix B.52 on Page 467.

Continued >>

Example: This example shows how to obtain the users registration data:

```
#include <ve_h323_prototypes.h>

/* Update the users registration data */
{
   t_ve_return_code_e            ret_code;
   t_ve_h323_user_registration_s ve_user_reg;

   /* Get VeriCall Edge's internal user registration data */
   ret_code = VE_H323GetUsersRegistrationData( applicationId,
                 &ve_user_handle,
                 &ve_user_reg );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Alter registration data */
   strncat((char*)(char*)user_reg.h323_user.h323_name_id.name,
               "192.168.1.102", VE_H323_MAX_NAME_ID_LEN);

   /* Update VeriCall Edge's internal data with the changed user
      registration data */
   ret_code = VE_H323UpdateUsersRegistrationData( applicationId,
               &ve_user_handle,
               &ve_user_reg );
   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */
}
```

VE_H323StartCall

Syntax:

```
t_ve_return_code_e  VE_H323StartCall(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_h323_dest_address_s *ve_h323_dest_address,
                    t_ve_h323_call_handle_s *ve_h323_call_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_h323_dest_address	Pointer to the structure containing the destination of the call to be initiated.
ve_h323_call_handle	Pointer to a structure to store the handle identifier of the call instance.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_H323StopCall**

Description: **VE_H323StartCall** is used to initiate a H.323 call to the specified destination.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The remote destination address for the call is contained in the structure pointed to by *ve_h323_dest_address*. This structure of type **t_ve_h323_dest_address_s** and is described in Appendix B.53 on Page 468.

The handle to identify this call instance is returned by this function and stored in the structure pointed to by *ve_h323_call_handle*. This structure of type **t_ve_h323_call_handle_s** is described in Appendix B.55 on Page 469. The handle should be used for all further VeriCall Edge API calls relating to this call instance.

Continued >>

Example: This simple example shows how to start a H.323 call:

```
#include <ve_h323_prototypes.h>

/* start a H.323 call */
{
   t_ve_return_code_e          ret_code;
   t_ve_h323_dest_address_s    ve_h323_dest_address;
   t_ve_h323_call_handle_s     ve_h323_call_handle;

   /* make sure the remote_uri structure is initialized */
   memset( &ve_h323_dest_address, '\0', sizeof(
t_ve_h323_dest_address_s ) );

   /* Set the URI of the person we wish to call */
   strcpy((char*)h323_dest_address.address, "TA:192.168.0.1");

   /* Place the new SIP call */
   ret_code = VE_H323StartCall(applicationId,
           &ve_user_handle,
           &ve_h323_dest_address,
           &ve_h323_call_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */
}
```

VE_H323StopCall

Syntax:

```
t_ve_return_code_e  VE_H323StopCall(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_h323_call_handle_s *ve_h323_call_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_h323_call_handle	Pointer to the structure containing the handle of the call instance.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_H323GetUsersRegistrationData**

Description: **VE_H323StopCall** is used to terminate a call instance.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify this call instance is contained in the structure pointed to by *ve_h323_call_handle* and was either returned through a call to **VE_H323StartCall** or obtained from the structure associated with the VE_USER_MSG_H323_CALL_STARTED message obtained via **VE_GetNextEvent**. The **t_ve_h323_call_handle_s** structure is described in Appendix B.55 on Page 469.

Note that either the initiator or the recipient of a call may use this function.

Continued >>

Example: This simple example shows how to stop a H.323 call:

```
#include <ve_h323_prototypes.h>

/* Stop a H.323 call */
{
   t_ve_return_code_e          ret_code;
   t_ve_h323_call_handle_s     ve_h323_call_handle

   /* Stop the call. */

   ret_code = VE_H323StopCall(applicationId,
          &ve_user_handle,
          &h323_call_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */
}
```

VE_H323UpdateCall

Syntax:

```
t_ve_return_code_e  VE_H323UpdateCall(
      UWord32 ve_application_id,
      t_ve_user_handle_s *ve_user_handle,
      t_ve_h323_call_update_user_channel_details_s *user_details,
      t_ve_h323_call_handle_s *ve_h323_call_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
user_details	Pointer to the structure containing the users channel details that are to be updated.
ve_h323_call_handle	Pointer to the structure containing the handle of the call instance.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_H323GetUsersRegistrationData**, **VE_H323StopCall**

Description: **VE_H323UpdateCall** is used to update a call instance. Updates to a call instance may be required to force a media change (including media attributes) or reduce/increase the bandwidth available to the call channel(s).

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The channel details to be updated are held within the structure pointed to by user_details. This structure, **t_ve_h323_call_update_user_channel_details_s** as described in Appendix B.56 on Page 470, contains the users channel(s) details and any modifications required to the call channel(s).

The handle to identify this call instance is contained in the structure pointed to by *ve_h323_call_handle* and was either returned through a call to **VE_H323StartCall** or obtained from the structure associated with the VE_USER_MSG_H323_CALL_STARTED message obtained via **VE_GetNextEvent**. The **t_ve_h323_call_handle_s** structure is described in Appendix B.55 on Page 469.

Continued >>

Example: This example shows how to update an active call:

```
#include <ve_h323_prototypes.h>

{
   t_ve_return_code_e                                ret_code;
   t_ve_h323_call_update_user_channel_details_s
                            call_update_user_channel_details;

   /* zero call_update_user_channel_details */
   memset(&call_update_user_channel_details, 0,
          sizeof(t_ve_h323_call_update_user_channel_details_s));

   /* Change the configuration */
   call_update_user_channel_details.
                     max_call_bandwidth_kbits_per_sec = 512;
   call_update_user_channel_details.number_of_channels = 2;
   call_update_user_channel_details.channel_association[0].
            call_update_attributes[0].media_id = G729_AB_MEDIA_ID;
   call_update_user_channel_details.channel_association[0].
            call_update_attributes[0].rtp_config.g729a.vad_state =
                                       VE_MEDIA_VAD_STATE_ENABLE;
   call_update_user_channel_details.channel_association[0].
       call_update_attributes[0].rtp_config.g729a.frame_size = 20;
   call_update_user_channel_details.channel_association[0].
            num_call_update_attributes = 1;
   call_update_user_channel_details.channel_association[1].
          call_update_attributes[0].media_id = H263_QCIF_MEDIA_ID;
   call_update_user_channel_details.channel_association[1].
            num_call_update_attributes = 1;

   /* update the H.323 call */
   if ((result = VE_H323UpdateCall(applicationId,
               &ve_user_handle,
               &call_update_user_channel_details,
               &h323_call_handle)) != VE_RETURN_CODE_SUCCESS)

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */
}
```

VE_H323CallResponse

Syntax:

```
t_ve_return_code_e  VE_H323CallResponse(
                UWord32 ve_application_id,
                t_ve_user_handle_s *ve_user_handle,
                t_ve_h323_call_handle_s *ve_h323_call_handle,
                t_ve_h323_call_response_e ve_h323_call_response,
                t_ve_h323_call_waiting_flag_e ve_call_waiting_flag )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user under consideration.
ve_h323_call_handle	Pointer to the structure containing the handle of the call instance under consideration.
ve_h323_call_response	A response code number.
ve_call_waiting_flag	A flag to indicate inclusion of call waiting in specific H.323 responses.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_GetNextEvent**

Description: **VE_H323CallResponse** is a H.323 call instance response to the initiator of a H.323 message received via **VE_GetNextEvent**. The recipient of the call may choose the appropriate response type to inform the caller whether the call will be accepted or rejected.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** and is described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify this call instance is contained in the structure pointed to by *ve_h323_call_handle* and was either returned through a call to **VE_H323StartCall** or obtained from the structure associated with the VE_USER_MSG_H323_CALL_STARTED message obtained via **VE_GetNextEvent**. The **t_ve_h323_call_handle_s** structure is described in Appendix B.55 on Page 469.

The H.323 response code is held in *ve_h323_call_response* and will indicate success, failure and external responses to H.323 specific function calls. The response code enumeration of type **t_ve_h323_call_response_e** is described in Appendix B.206 on Page 626.

Continued >>

Setting the *ve_call_waiting_flag* to be true when the response is of type `Alerting`, will result in the inclusion of the call waiting invoke command in the Alerting response. This will place the destination of the command response in call waiting. The enumeration type **t_ve_h323_call_waiting_flag_e** is described in Appendix B.216 on Page 642.

Example: This simple example shows how to generate a H.323 response:

```
#include <ve_h323_prototypes.h>

/* Respond to an incoming call using VE_H323CallResponse */
{
   t_ve_return_code_e     ret_code;

   /* Assume our handset is not busy, set H.323 state to ringing */
   ret_code = VE_H323CallResponse(applicationId,
                        &ve_user_handle,
                        &ve_h323_call_handle,
                        VE_H323_RESPONSE_ALERTING,
                        VE_H323_CALL_WAITING_FLAG_FALSE);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */
}
```

VE_H323SendUserInput

Syntax:

```
t_ve_return_code_e  VE_H323SendUserInput(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_h323_call_handle_s *ve_h323_call_handle,
                    Word8 ve_digit )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_h323_call_handle	Pointer to a structure to store the handle identifier of the call instance.
ve_digit	The character to be sent via the H.323 control plane.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_H323SendUserInput** is used to send a user defined character via the H.323 control plane. This function is designed for transmitting out of band DTMF relay and, for example, can be called when a user presses a function key.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify this call instance is contained in the structure pointed to by *ve_h323_call_handle* and was either returned through a call to **VE_H323StartCall** or obtained from the structure associated with the VE_USER_MSG_H323_CALL_STARTED message obtained via **VE_GetNextEvent**. The **t_ve_h323_call_handle_s** structure is described in Appendix B.55 on Page 469.

The user-defined character is passed to the function via *ve_digit*.

Continued >>

Example: This simple example shows how to send a digit:

```
#include <ve_h323_prototypes.h>

/* Send a digit */
{
   /* Assume: ve_user_handle has been stored previously */
   t_ve_return_code_e          ret_code;

   /* Send the digit */
   ret_code = VE_H323SendUserInput( applicationId,
          &ve_user_handle,
          &h323_call_handle,
          '1' );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect an event */
}
```

VE_H323VideoFastUpdate

Syntax:

```
t_ve_return_code_e  VE_H323VideoFastUpdate(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_h323_call_handle_s *ve_h323_call_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_h323_call_handle	Pointer to the structure containing the call instance handle.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_H323VideoFreeze**

Description: **VE_H323VideoFastUpdate** allows the VeriCall Edge User Application to request a video image update. This would be done, for example, if the video stream had been previously frozen with a call to **VE_H323VideoFreeze**. This API function call would request the far-end to update the video image with an I-Frame.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify this call instance is contained in the structure pointed to by *ve_h323_call_handle* and was either returned through a call to **VE_H323StartCall** or obtained from the structure associated with the VE_USER_MSG_H323_CALL_STARTED message obtained via **VE_GetNextEvent**. The **t_ve_h323_call_handle_s** structure is described in Appendix B.55 on Page 469.

Continued >>

Example: This simple example shows how to request a video fast update:

```
#include <ve_h323_prototypes.h>

{
   /* Assume: ve_user_handle has been stored previously */
   t_ve_return_code_e          ret_code;

   /* Request a video fast update */
   ret_code = VE_H323VideoFastUpdate( applicationId,
          &ve_user_handle,
          &h323_call_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect an event */
}
```

VE_H323VideoFreeze

Syntax:

```
t_ve_return_code_e  VE_H323VideoFreeze(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_h323_call_handle_s *ve_h323_call_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_h323_call_handle	Pointer to the structure containing the call instance handle.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_H323VideoFastUpdate**

Description: **VE_H323VideoFreeze** allows the VeriCall Edge User Application to send a request to the far-end for the video stream to be frozen. This is similar to a remote pause control, in that the image stream freezes on and displays only a single frame.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The handle to identify this call instance is contained in the structure pointed to by *ve_h323_call_handle* and was either returned through a call to **VE_H323StartCall** or obtained from the structure associated with the VE_USER_MSG_H323_CALL_STARTED message obtained via **VE_GetNextEvent**. The **t_ve_h323_call_handle_s** structure is described in Appendix B.55 on Page 469.

Continued >>

Example: This simple example shows how to request a video freeze:

```
#include <ve_h323_prototypes.h>

{
   /* Assume: ve_user_handle has been stored previously */
   t_ve_return_code_e          ret_code;

   /* Request a video freeze */
   ret_code = VE_H323VideoFreeze( applicationId,
          &ve_user_handle,
          &h323_call_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect an event */
}
```

VE_H323CallTransferConsult

Syntax:

```
t_ve_return_code_e  VE_H323CallTransferConsult(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_h323_call_handle_s *ve_h323_call_handle1,
                    t_ve_h323_call_handle_s *ve_h323_call_handle2 )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_h323_call_handle1	Pointer to the structure containing the primary call instance handle.
ve_h323_call_handle2	Pointer to the structure containing the secondary call instance handle.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_H323CallTransferDontConsult**

Description: **VE_H323CallTransferConsult** allows the VeriCall Edge User Application to transfer a call from one instance to another call instance with consultation.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The separate call instances are identified by *ve_h323_call_handle1*, for the primary call instance, and *ve_h323_call_handle1* for the secondary call instance. These structures of type **t_ve_h323_call_handle_s** are defined in **ve_h323_defs.h** and are described in Appendix B.55 on Page 469. These primary and secondary call handles must be those returned from a **VE_H323StartCall** function call.

Continued >>

Example: This example shows how to transfer a call with consultation:

```
#include <ve_h323_prototypes.h>

/* Start a call transfer with consultation (attended) */
{
   t_ve_return_code_e          ret_code;
   t_ve_user_handle_s          ve_user_handle:
   t_ve_h323_call_handle_s     ve_h323_call_handle2;

   /* Start a call transfer with consultation (attended) */
   ret_code = VE_H323CallTransferConsult(applicationId,
               &ve_user_handle,
               &ve_h323_call_handle1,
               &ve_h323_call_handle2 );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */
}
```

VE_H323CallTransferDontConsult

Syntax:

```
t_ve_return_code_e  VE_H323CallTransferDontConsult(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_h323_call_handle_s *ve_h323_call_handle,
                    t_ve_h323_dest_address_s *ve_h323_dest_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_h323_call_handle	Pointer to the structure containing the primary call instance handle.
ve_h323_dest_handle	Pointer to the structure containing the secondary call destination address.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_H323CallTransferConsult**

Description: **VE_H323CallTransferDontConsult** allows the VeriCall Edge User Application to transfer a call from one instance to another call address.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The primary call instance is identified by *ve_h323_call_handle*, which is of structure type **t_ve_h323_call_handle_s** is described in Appendix B.55 on Page 469. The primary call handle must be that returned from a **VE_H323StartCall** function call.

The destination address that the call is to be transferred to is identified by *ve_h323_dest_address*, which is a structure of type **t_ve_h323_dest_address_s**, as defined in Appendix B.53 on Page 468.

Continued >>

Example: This example shows how to transfer a call without consultation:

```
#include <ve_h323_prototypes.h>

/* Start a call transfer without consultation (blind) */
{
   t_ve_return_code_e          ret_code;
   t_ve_h323_dest_address_s    ve_h323_dest_address;

   /* Setup the target */
   memset(ve_h323_dest_address, 0 ,
sizeof(t_ve_h323_dest_address_s));

   strncpy( ( char* )ve_h323_dest_address.address, "TEL:5351",
     VE_H323_MAX_ADDRESS_LEN );

   /* Start a call transfer with consultation (attended) */
   ret_code = VE_H323CallTransferDontConsult(applicationId,
               &ve_user_handle,
               &ve_h323_call_handle,
               &ve_h323_dest_address );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */
}
```

VE_H323CallForward

Syntax:

```
t_ve_return_code_e  VE_H323CallForward(
  UWord32 ve_application_id,
  t_ve_user_handle_s ve_user_handle,
  t_ve_h323_call_handle_s *ve_h323_call_handle,
  t_ve_h323_dest_address_s *ve_h323_dest_address,
  t_ve_h323_call_forward_reason_e ve_h323_call_forward_reason )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_h323_call_handle	Pointer to the structure containing the call instance handle.
ve_h323_dest_address	Pointer to the structure containing the destination address to forward this call instance to.
ve_h323_call_forward_reason	Enumerated reason for forwarding this call instance.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_H323CallForwardActivate, VE_H323CallForwardDeactivate**

Description: **VE_H323CallForward** allows the VeriCall Edge User Application to forward a call to an alternative destination address.

The *ve_application_id* argument must be a valid registered application identifier. The user identification handle, obtained when the user was added to the VeriCall Edge system using the **VE_AddUser** function, is contained in the structure pointed to by *ve_user_handle*. This structure of type **t_ve_user_handle_s**, is described in Appendix B.25 on Page 299. The call instance to be forwarded is identified by *ve_h323_call_handle*. This structure is of type **t_ve_h323_call_handle_s** as defined in **ve_h323_defs.h** and is described in Appendix B.55 on Page 469. This call handle must be that returned from a **VE_H323StartCall** function call.

The destination to forward the call to is identified by *ve_h323_dest_address* argument, which is a pointer to the **t_ve_h323_dest_address_s** structure, defined in Appendix B.53 on Page 468. The reason for this call instance being forwarded is identified by the *ve_h323_call_forward_reason* enumerated type, **t_ve_h323_call_forward_reason_e** as defined in Appendix B.205 on Page 625.

Continued >>

Example: This example shows how to forward a call:

```
#include <ve_h323_prototypes.h>

/* Forward a call */
{
   t_ve_return_code_e          ret_code;
   t_ve_h323_dest_address_s    ve_h323_dest_address;

   /* Our handset is not busy to set H323 state to ringing */
   result = VE_H323CallResponse( applicationId,
                          &ve_user_handle,
                          &ve_h323_call_handle,
                          VE_H323_RESPONSE_ALERTING,
                          VE_H323_CALL_WAITING_FLAG_FALSE);

   /* The address we will be forwarding to */
   strncpy((char*)ve_h323_dest_address.address, "TEL:5352",
                          VE_H323_MAX_ADDRESS_LEN);

   /* Forward the call */
   ret_code = VE_H323CallForward( applicationId,
                                  &ve_user_handle,
                                  &ve_h323_call_handle,
                                  &ve_h323_dest_address,
                                  VE_H323_CALL_FORWARD_TYPE_BUSY);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */
}
```

VE_H323CallForwardActivate

Syntax:

```
t_ve_return_code_e  VE_H323CallForwardActivate(
  UWord32 ve_application_id,
  t_ve_user_handle_s *ve_user_handle,
  t_ve_h323_call_handle_s *ve_h323_call_handle,
  t_ve_h323_dest_address_s *server_addr,
  t_ve_h323_dest_address_s *ve_h323_dest_address,
  t_ve_h323_call_forward_reason_e ve_h323_call_forward_reason )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_h323_call_handle	Pointer to the structure containing the call instance handle.
server_addr	Pointer to the structure containing the address of the server supporting the call forwarding service.
ve_h323_dest_address	Pointer to the structure containing the destination address to forward this call instance to.
ve_h323_call_forward_reason	Enumerated reason for forwarding this call instance.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_H323CallForwardDeactivate, VE_H323CallForward**

Description: **VE_H323CallForwardActivate** allows the activation of a call diversion service on the specified remote server.

The *ve_application_id* argument must be a valid registered application identifier. The user identification handle, obtained when the user was added to the VeriCall Edge system using the **VE_AddUser** function, is contained in the structure pointed to by *ve_user_handle*. This structure of type **t_ve_user_handle_s**, is described in Appendix B.25 on Page 299. The call instance for which call forwarding is to be activated is identified by *ve_h323_call_handle*. This structure is of type **t_ve_h323_call_handle_s** as defined in **ve_h323_defs.h** and is described in Appendix B.55 on Page 469. This call handle must be that returned from a **VE_H323StartCall** function call.

Continued >>

The address of the server, which offers the call forwarding service, is identified by *server_addr*. The destination address to forward the call to is identified by *ve_h323_dest_address* argument. These arguments are pointers to the **t_ve_h323_dest_address_s** structure, defined in Appendix B.53 on Page 468. The reason for this users call instances being forwarded is identified by the *ve_h323_call_forward_reason* enumerated type, **t_ve_h323_call_forward_reason_e** as defined in Appendix B.205 on Page 625.

Example: This example shows how to activate call forwarding:

```
#include <ve_h323_prototypes.h>

/* Activate call forwarding */
{
   t_ve_return_code_e           ret_code;
   t_ve_h323_dest_address_s     ve_h323_dest_address;
   t_ve_h323_dest_address_s     ve_h323_call_forward_server_addr;
   t_ve_h323_default_configuration_s    h323_config

   /* Get the default H.323 stack configuration */
   ret_code = VE_GetSystemAttribute(applicationId,
                    VE_ATTRIBUTE_TYPE_DEFAULT_H323_CONFIGURATION,
                    VE_SINGLETON_DATA_ENTITY,
                    &h323_config,
                    sizeof (t_ve_h323_default_configuration_s));

   /* Address to forward calls to */
   strncpy((char*)ve_h323_dest_address.address, "TEL:5353",
                       VE_H323_MAX_ADDRESS_LEN);

   /* Address of the server to activate call forwarding on */
   strncpy((char*)ve_h323_call_forward_server_addr.address,
      (const char *)h323_config.ras.gatekeeper_primary.address,
                                   VE_H323_MAX_ADDRESS_LEN);

   /* Activate call forwarding */
   ret_code = VE_H323CallForwardActivate( applicationId,
                        &ve_user_handle,
                        &ve_h323_call_handle,
                        &ve_h323_call_forward_server_addr,
                        &ve_h323_dest_address,
                        VE_H323_CALL_FORWARD_TYPE_UNCONDITIONAL);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */
}
```

VE_H323CallForwardDeactivate

Syntax:

```
t_ve_return_code_e  VE_H323CallForwardDeactivate(
  UWord32 ve_application_id,
  t_ve_user_handle_s *ve_user_handle,
  t_ve_h323_call_handle_s *ve_h323_call_handle,
  t_ve_h323_dest_address_s *server_addr,
  t_ve_h323_call_forward_reason_e ve_h323_call_forward_reason )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_h323_call_handle	Pointer to the structure containing the call instance handle.
server_addr	Pointer to the structure containing the address of the server supporting the call forwarding service.
ve_h323_call_forward_reason	Enumerated reason for forwarding this call instance.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_H323CallForwardActivate, VE_H323CallForward**

Description: **VE_H323CallForwardDeactivate** allows the de-activation of a call diversion service on the specified server.

The *ve_application_id* argument must be a valid registered application identifier. The user identification handle, obtained when the user was added to the VeriCall Edge system using the **VE_AddUser** function, is contained in the structure pointed to by *ve_user_handle*. This structure of type **t_ve_user_handle_s**, is described in Appendix B.25 on Page 299. The call instance for which call forwarding is to be deactivated is identified by *ve_h323_call_handle*. This structure is of type **t_ve_h323_call_handle_s** as defined in **ve_h323_defs.h** and is described in Appendix B.55 on Page 469. This call handle must be that returned from a **VE_H323StartCall** function call.

The address of the server, which offers the call forwarding service, is identified by *server_addr* and is a pointer to the **t_ve_h323_dest_address_s** structure, defined in Appendix B.53 on Page 468. The reason for this users call forwarding being deactivated is identified by the *ve_h323_call_forward_reason* enumerated type, **t_ve_h323_call_forward_reason_e** as defined in Appendix B.205 on Page 625.

Continued >>

Example: This example shows how to de-activate call forwarding:

```
#include <ve_h323_prototypes.h>

/* De-activate call forwarding */
{
   t_ve_return_code_e            ret_code;
   t_ve_h323_dest_address_s      ve_h323_dest_address;
   t_ve_h323_dest_address_s      ve_h323_call_forward_server_addr;
   t_ve_h323_default_configuration_s    h323_config

   /* Get the default H.323 stack configuration */
   result = VE_GetSystemAttribute(applicationId,
                     VE_ATTRIBUTE_TYPE_DEFAULT_H323_CONFIGURATION,
                     VE_SINGLETON_DATA_ENTITY,
                     &h323_config,
                     sizeof (t_ve_h323_default_configuration_s));

   /* Address of the server to de-activate call forwarding on */
   strncpy((char*)ve_h323_call_forward_server_addr.address,
      (const char *)h323_config.ras.gatekeeper_primary.address,
                                 VE_H323_MAX_ADDRESS_LEN);

   /* Activate call forwarding */
   ret_code = VE_H323CallForwardDeactivate( applicationId,
                        &ve_user_handle,
                        &ve_h323_call_handle,
                        &ve_h323_call_forward_server_addr,
                        VE_H323_CALL_FORWARD_TYPE_UNCONDITIONAL);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Expect a success or failure event */
}
```

VE_H323CallHoldOn

Syntax:

```
t_ve_return_code_e  VE_H323CallHoldOn(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_h323_call_handle_s *ve_h323_call_handle,
                    t_ve_h323_hold_method_e ve_h323_hold_method )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_h323_call_handle	Pointer to the structure containing the call instance handle.
ve_h323_hold_mthod	Enumerated type that determines the call hold method to be used.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_H323CallHoldOff**

Description: **VE_H323CallHoldOn** allows the VeriCall Edge User Application to place the referenced call on hold.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The call instance to be placed on hold is identified by *ve_h323_call_handle*. This structure is of type **t_ve_h323_call_handle_s** as defined in **ve_h323_defs.h** and is described in Appendix B.55 on Page 469. This call handle must be that returned from a **VE_H323StartCall** function call.

The method employed by this function to place the call on hold is determined by the *ve_h323_hold_method_e* argument. This is an enumerated type, **t_ve_h323_hold_method_e**, as described in Appendix B.221 on Page 647.

Continued >>

Example: This example shows how to put a H.323 call on hold:

```
#include <ve_h323_prototypes.h>

{
   t_ve_return_code_e     ret_code;

   /* Instruct the Call to go on hold using the H.450.4 method */
   ret_code = VE_H323CallHoldOn( applicationId,
            &ve_user_handle,
            &ve_h323_call_handle,
            VE_H323_HOLD_METHOD_H450_4 );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect an event */

   /* If the event stated that the hold was successful then the
      media will have been stopped */

   /* Now go off hold */
   ret_code = VE_H323CallHoldOff( applicationId,
             &ve_user_handle,
             &ve_h323_call_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect an event */
}
```

VE_H323CallHoldOff

Syntax:

```
t_ve_return_code_e  VE_H323CallHoldOff(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s *ve_user_handle,
                    t_ve_h323_call_handle_s *ve_h323_call_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
ve_h323_call_handle	Pointer to the structure containing the call instance handle.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: **VE_H323CallHoldOn**

Description: **VE_H323CallHoldOff** allows the VeriCall Edge User Application to take the referenced call off hold.

The *ve_application_id* argument must be a valid registered application identifier. The handle pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

The call instance to be taken off hold is identified by *ve_h323_call_handle*. This structure is of type **t_ve_h323_call_handle_s** as defined in **ve_h323_defs.h** and is described in Appendix B.55 on Page 469. This call handle must be that returned from a **VE_H323StartCall** function call.

Continued >>

Example: This example shows how to take a H.323 call off hold:

```
#include <ve_h323_prototypes.h>

{
   t_ve_return_code_e     ret_code;

   /* Instruct the Call to go on hold using the H.450.4 method */
   ret_code = VE_H323CallHoldOn( applicationId,
            &ve_user_handle,
            &ve_h323_call_handle,
            VE_H323_HOLD_METHOD_H450_4 );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect an event */

   /* If the event stated that the hold was successful then the
      media will have been stopped */

   /* Now go off hold */
   ret_code = VE_H323CallHoldOff( applicationId,
             &ve_user_handle,
             &ve_h323_call_handle );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Expect an event */
}
```

VE_H323MWIInterrogate

Syntax:

```
t_ve_return_code_e  VE_H323MWIInterrogate(
                    UWord32 ve_application_id,
                    t_ve_user_handle_s ve_user_handle,
                    t_ve_h323_dest_address_s *message_center_addr,
                    Word8 *service )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.
message_center_addr	Pointer to the structure containing the message center address to send the de-activation request to.
service	Enumerated message waiting service type.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_H323MWIInterrogate** allows the VeriCall Edge User Application to send an interrogation request to an external Message Center. The User Application is termed the Served User.

Normally a H.323 Message Waiting Indication (MWI) is sent to the User Application through the VE_USER_MSG_H323_MESSAGE_WAITING event, described in Appendix B.162 on Page 584. The **VE_H323MWIInterrogate** function allows the User Application to query the Message Center, whose address is identified by the *message_center_addr* argument, which is of the **t_ve_h323_dest_address_s** structure type as described in Appendix B.53 on Page 468.

The use of this function is dependent on the Message Center application. It is envisioned that the use of this function will cause the Message Center to respond with the aforementioned MWI event.

The *ve_application_id* argument must be a valid registered application identifier. The user identification handle, obtained when the user was added to the VeriCall Edge system using the **VE_AddUser** function, is contained in the structure pointed to by *ve_user_handle*.

The *service* argument indicates to the Message Center the type of service notification that is being queried. The type and indicating number of the service is dependent on the Message Center application.

Continued >>

Example: This simple example shows how to interrogate a MW server:

```
#include <ve_h323_prototypes.h>

/* Interrogate a Message Waiting server */
{
   t_ve_return_code_e          ret_code;
   t_ve_h323_dest_address_s    message_center_addr;

   /* Address of the message center */
   strncpy((char*)message_center_addr.address, "TEL5354",
                                 VE_H323_MAX_ADDRESS_LEN);

   /* For more information on the service, see either the
      documentation that accompanies the message centre software or
      examine the MWI ans.1 spec */

   ret_code = VE_H323MWIInterrogate( applicationId,
                  &ve_user_handle,
                  &message_center_addr,
                  (Word8*)"speech" );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   /* Now expect an event with the MWI status */
}
```

Chapter 9

VeriCall Edge™ NAT API

This chapter describes the VeriCall Edge Network Address Translation (NAT) Traversal Module API functions. The VeriCall Edge NAT API is used for the configuration and control of the NAT Traversal Module, allowing the VeriCall Edge system to traverse the various types of NAT for both SIP signaling and media streams.

9.1 VeriCall Edge NAT API Function Descriptions

For each function detailed in this section, the following are described (as appropriate):

- **Syntax** Command syntax.
- **Arguments** Description of the arguments.
- **Returns** Return codes from the function.
- **See Also** Other functions that perform related tasks.
- **Description** What the function can be used for.
- **Example** Short code sections showing how to use the function.

9.2 VeriCall Edge NAT API Functions

The VeriCall Edge NAT API functions are described on the following pages:

Function	Page
VE_NATStartDiscover	227
VE_NATSetDiscoverData	229
VE_NATGetDiscoverData	230
VE_NATCancelDiscover	231
VE_NATReserveUserAddresses	232
VE_NATSetChannelAddressesData	234
VE_NATGetChannelAddressesData	236
VE_NATReleaseUserAddresses	238
VE_NATStartKeepAlive	240
VE_NATStopKeepAlive	242

VE_NATStartDiscover

Syntax:

```
t_ve_return_code_e  VE_NATStartDiscover(
                        UWord32 ve_application_id,
                        t_ve_nat_init_s *ve_nat_init )
```

Arguments:

ve_application_id	The registered application ID.
ve_nat_init	Pointer to a structure containing the discover method to be used as well as the configuration of STUN.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_NATStartDiscover** is used to initiate a NAT discovery operation and should be called soon after the VeriCall Edge system has been committed, via **VE_Commit**, to ensure the correct NAT settings are applied to the system.

Use of this function will result in a VE_USER_MSG_NAT_DISCOVER_RESPONSE event containing the data that can be applied using **VE_NATSetDiscoverData**. A second event will also be generated when the NAT binding lifetime is discovered.

Any failure during the discovery operation will be reported via a VE_USER_MSG_NAT_REQUEST_FAILED event.

The ve_application_id argument must be a valid registered application identifier.

The discover method to be used is pointed to by the *ve_nat_init* argument which is a pointer to the **t_ve_nat_init_s** structure, described in Appendix B.62 on Page 494.

A discovery operation may be cancelled at any time using VE_NATCancelDiscover.

A NAT discovery operation should only be started during "idle periods" - defined as when no other discovery operation is in progress and no channel address bindings are in use. Attempts to start a discovery operation at any other time will result in a failure event.

Continued >>

Example: This example shows how to start NAT discovery:

```
#include <ve_nat_prototypes.h>

/* Start a NAT discovery background task using RFC3489 method
   using a STUN server located at larry.gloo.net:3478 */
{
   t_ve_nat_init_s nat_init;
   t_ve_return_code_e ret_code;

   nat_init.discover_method = VE_NAT_DISCOVER_METHOD_RFC3489;
   nat_init.stun.state = VE_NAT_STATE_ENABLED;
   nat_init.stun.ip.ip_configuration = VE_MEDIA_IP_CONFIG_FQDN;
   strncpy(&stun_ip.ip_specification.ip_domain[0],
               "larry.gloo.net", VE_MAX_MEDIA_IP_ADDRESS_SIZE);
   nat_init.stun.port  = 3478;
   nat_init.turn.state  = VE_NAT_STATE_DISABLED;

   ret_code = VE_NATStartDiscover(application_id, &nat_init);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Discovery data in resulting VE_USER_MSG_NAT_DISCOVER_RESPONSE
      events should be applied using VE_NATSetDiscoverData() */
}
```

VE_NATSetDiscoverData

Syntax:

```
t_ve_return_code_e  VE_NATSetDiscoverData(
                        UWord32 ve_application_id,
                        t_ve_nat_configuration_s *data )
```

Arguments:

ve_application_id	The registered application ID.
data	Pointer to a structure containing the NAT discover data.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_NATSetDiscoverData** applies the NAT device configuration to the VeriCall Edge system. The NAT device configuration can be constructed manually or be obtained from a VE_USER_MSG_NAT_DISCOVER_RESPONSE event as a result of calling **VE_NATStartDiscover**.

The ve_application_id argument must be a valid registered application identifier.

The NAT discover data is contained in the structure pointed to by the *data* argument. This structure, **t_ve_nat_configuration_s**, is described in Appendix B.63 on Page 496.

Example: This example shows how to set the NAT discovered data:

```
#include <ve_nat_prototypes.h>

/* Extract NAT device discovery data from
   VE_USER_MSG_NAT_DISCOVER_RESPONSE event <event> and apply to the
   VeriCall Edge system */
{
   t_ve_nat_configuration_s *config =
                     &event->message.nat_discover_response.config;
   t_ve_return_code_e ret_code;

   ret_code = VE_NATSetDiscoverData(application_id, config);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_NATGetDiscoverData

Syntax:

```
t_ve_return_code_e  VE_NATGetDiscoverData(
                        UWord32 ve_application_id,
                        t_ve_nat_configuration_s *data )
```

Arguments:

ve_application_id	The registered application ID.
data	Pointer to a structure that will contain the NAT discover data.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_NATGetDiscoverData** is used to obtain the NAT device configuration to the VeriCall Edge system. This function checks the parameters obtained from the VeriCall Edge attribute.

The ve_application_id argument must be a valid registered application identifier.

The NAT discover data is stored in the structure pointed to by the *data* argument. This structure, **t_ve_nat_configuration_s**, is described in Appendix B.63 on Page 496.

Example: This example shows how to obtain the current NAT discovered data:

```
#include <ve_nat_prototypes.h>

/* Get NAT device configuration data currently used by the VeriCall
   Edge system */
{
   t_ve_nat_configuration_s config;
   t_ve_return_code_e ret_code;

   ret_code = VE_NATGetDiscoverData(application_id, &config);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   else
   {
      /* Process data in config */
   }
}
```

VE_NATCancelDiscover

Syntax: **`t_ve_return_code_e VE_NATCancelDiscover(`**
`UWord32` *`ve_application_id`* **`)`**

Arguments: *`ve_application_id`* The registered application ID.

Returns: An enumerated return code indicating success (`VE_RETURN_CODE_SUCCESS`) or an error code. The returned enumeration type, **`t_ve_return_code_e`**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_NATCancelDiscover** is used to cancel the current NAT discovery operation previously started using **VE_NATStartDiscover**.

The cancelled operation will result in a `VE_USER_MSG_NAT_REQUEST_FAILED` event occurring. It should be noted that there is a possibility that the discover operation completes before the cancel request is acted upon in which case the resulting `VE_USER_MSG_NAT_DISCOVER_RESPONSE` event can simply be ignored.

The ve_application_id argument must be a valid registered application identifier.

Example: This example shows how to cancel NAT discovery:

```
#include <ve_nat_prototypes.h>

/* Cancel current NAT discovery background task */
{
   t_ve_return_code_e ret_code;

   ret_code = VE_NATCancelDiscover(application_id);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_NATReserveUserAddresses

Syntax:

```
t_ve_return_code_e  VE_NATReserveUserAddresses(
                        UWord32 ve_application_id,
                        t_ve_user_handle_s *ve_user_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_NATReserveUserAddresses** initiates a NAT binding operation to provide public address mappings for each of the user's channels. The operation will result in a VE_USER_MSG_NAT_USER_ADDRESSES_RESPONSE event for each channel containing data that must be applied using **VE_NATSetChannelAddressesData**. Any failure during the binding operation for each channel will be reported via a VE_USER_MSG_NAT_REQUEST_FAILED event. Public address bindings for a user will remain active until **VE_NATReleaseUserAddresses** is called.

The ve_application_id argument must be a valid registered application identifier.

The user's handle, against which the addresses are to be reserved, pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

NAT binding operations should only be attempted after a call has been made to VE_NATStartDiscover.

Continued >>

Example: This example shows how to reserve the NAT discovered data:

```
#include <ve_nat_prototypes.h>

/* Obtain public addresses for user's channels */
{
   t_ve_return_code_e ret_code;

   ret_code = VE_NATReserveUserAddresses(application_id,
                                         &user_handle);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Public address data in resulting
      VE_USER_MSG_NAT_USER_ADDRESSES_RESPONSE event(s) and should
      be applied using VE_NATSetChannelAddressesData() */
}
```

VE_NATSetChannelAddressesData

Syntax:

```
t_ve_return_code_e  VE_NATSetChannelAddressesData(
                     UWord32 ve_application_id,
                     UWord32 channel_number,
                     t_ve_user_sub_channel_select_e sub_channel,
                     t_ve_nat_channel_addresses_s *data )
```

Arguments:

ve_application_id	The registered application ID.
channel_number	The channel number to which these addresses are applicable to.
sub_channel	Enumerated type indicating the sub-channel selection.
data	Pointer to a structure containing the public address data.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_NATSetChannelAddressesData** applies the user channel's public address data (taking into account any NAT device) to the VeriCall system. This public address data can be constructed manually or be obtained from a VE_USER_MSG_NAT_USER_ADDRESSES_RESPONSE event as a result of calling **VE_NATReserveUserAddresses**.

The ve_application_id argument must be a valid registered application identifier.

The channel number against which these addresses are to be applied is identified by *channel_number*, while the sub-channel is selected by the *sub_channel* argument, an enumerated type of **t_ve_user_sub_channel_select_e** described in Appendix B.167 on Page 593.

The public address data to be stored in the VeriCall Edge attribute is contained in the **t_ve_nat_channel_addresses_s** structures, pointed to by the *data* argument. This structure is described in Appendix B.61 on Page 492.

Continued >>

Example: This example shows how to set the address data:

```
#include <ve_nat_prototypes.h>

/* Extract public address details for user channel from
   VE_USER_MSG_NAT_USER_ADDRESSES_RESPONSE event <event> and apply
   to the VeriCall Edge system */
{
   t_ve_return_code_e ret_code;
   UWord32 channel_number =
        event->message.nat_user_addresses_response.channel_number;
   t_ve_user_sub_channel_select_e sub_channel =
        event->message.nat_user_addresses_response.sub_channel;
   t_ve_nat_channel_addresses_s *data =
        &event->message.nat_user_addresses_response.chan_addr;

   ret_code = VE_NATSetChannelAddressesData(application_id,
                                   channel_number, sub_channel, data);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
}
```

VE_NATGetChannelAddressesData

Syntax:

```
t_ve_return_code_e  VE_NATGetChannelAddressesData(
                     UWord32 ve_application_id,
                     UWord32 channel_number,
                     t_ve_user_sub_channel_select_e sub_channel,
                     t_ve_nat_channel_addresses_s *data )
```

Arguments:

ve_application_id	The registered application ID.
channel_number	The channel number to which these addresses are applicable to.
sub_channel	Enumerated type indicating the sub-channel selection.
data	Pointer to a structure containing the public address data.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_NATGetChannelAddressesData** is used to obtain the channel address data of the VeriCall Edge attribute. This function checks the parameters obtained from the VeriCall Edge attribute.

The ve_application_id argument must be a valid registered application identifier.

The channel number against which these addresses are to be obtained is identified by *channel_number*, while the sub-channel is selected by the *sub_channel* argument, an enumerated type of **t_ve_user_sub_channel_select_e** described in Appendix B.167 on Page 593.

The public address data obtained from the VeriCall Edge attribute is stored in the **t_ve_nat_channel_addresses_s** structures, pointed to by the *data* argument. This structure is described in Appendix B.61 on Page 492.

Continued >>

Example: This example is to obtain the address data:

```
#include <ve_nat_prototypes.h>

/* Get public address details for channel 1 (primary sub-channel)
   currently used by the VeriCall Edge system */
{
   t_ve_nat_channel_addresses_s data;
   t_ve_return_code_e ret_code;

   ret_code = VE_NATGetChannelAddressesData(application_id,
                1,
                VE_USER_SUB_CHANNEL_SELECT_PRIMARY,
                &data );

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }
   else
   {
      /* Process data in data */
   }
}
```

VE_NATReleaseUserAddresses

Syntax:
```
t_ve_return_code_e  VE_NATReleaseUserAddresses(
                        UWord32 ve_application_id,
                        t_ve_user_handle_s *ve_user_handle )
```

Arguments:

ve_application_id	The registered application ID.
ve_user_handle	Pointer to the structure containing the handle of the user.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_NATReleaseUserAddresses** releases public address bindings that have been previously reserved for each of the specified user's channels using VE_NATReserveUserAddresses. This operation will result in a VE_USER_MSG_NAT_USER_ADDRESSES_RESPONSE event for each channel containing cleared (disabled) bindings data that must be applied using **VE_NATSetChannelAddressesData**.

Any failure during the release operation for each channel will be reported via a VE_USER_MSG_NAT_REQUEST_FAILED event.

The ve_application_id argument must be a valid registered application identifier.

The user's handle, against which the public addresses are to be released, pointed to by *ve_user_handle*, of structure type **t_ve_user_handle_s** described in Appendix B.25 on Page 299, must be that returned from a **VE_AddUser** function call.

Continued >>

Example: This example shows how to release the user addresses:

```
#include <ve_nat_prototypes.h>

/* Free public addresses for user's channels */
{
   t_ve_return_code_e ret_code;

   ret_code = VE_NATReleaseUserAddresses(application_id,
&user_handle);

   if(ret_code != VE_RETURN_CODE_SUCCESS)
   {
      /* Handle error */
   }

   /* Data in resulting VE_USER_MSG_NAT_USER_ADDRESSES_RESPONSE
      event(s) should be applied using
      VE_NATSetChannelAddressesData() */
}
```

VE_NATStartKeepAlive

Syntax:
```
t_ve_return_code_e  VE_NATStartKeepAlive(
                    UWord32 ve_application_id,
                    t_ve_network_connection_s *network_connection )
```

Arguments:

ve_application_id	The registered application ID.
network_connection	A pointer to a structure containing the network connection configuration parameters.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_NATStartKeepAlive** initiates a keep-alive operation for the specified network connection. The function returns immediately and, if successful, the NAT traversal module will attempt to start a keep-alive operation on the specified network connection. Once the keep-alive operation is started, events will be reported via VE_USER_MSG_NAT_KEEP_ALIVE_EVENT events. Any failure to start the keep-alive operation will be reported via a VE_USER_MSG_NAT_REQUEST_FAILED event.

The keep-alive operation may be stopped using **VE_NATStopKeepAlive**.

The ve_application_id argument must be a valid registered application identifier.

The network connection structure is pointed to by *network_connection* and can be copied directly from a VE_USER_MSG_NETWORK_CONNECTION_EVENT indicating a new open connection (recommended), or it can be set manually. This structure is of type **t_ve_network_connection_s** and is described in Appendix B.64 on Page 498.

NAT binding keep-alive operations should only be attempted after a call has been made to VE_NATStartDiscover.

Continued >>

Example: This example shows how to start a NAT keep-alive operation:

```
#include <ve_nat_prototypes.h>

/* Start keep-alive mechanism for new network connection reported
   by the VeriCall Edge system via a
   VE_USER_MSG_NETWORK_CONNECTION_EVENT event <event> */
{
   t_ve_return_code_e ret_code;
   t_ve_network_connection_s *network_connection =
               &event.message.network_connection_event.connection;

   if(event->message.network_connection_event.event ==
                                VE_NETWORK_CONNECTION_EVENT_OPENED)
   {
      ret_code = VE_NATStartKeepAlive( application_id,
                                       network_connection);

      if(ret_code != VE_RETURN_CODE_SUCCESS)
      {
          /* Handle error */
      }
   }
}
```

VE_NATStopKeepAlive

Syntax:

```
t_ve_return_code_e  VE_NATStopKeepAlive(
                    UWord32 ve_application_id,
                    t_ve_network_connection_s *network_connection )
```

Arguments:

ve_application_id	The registered application ID.
n*etwork_connection*	A pointer to a structure containing the network connection configuration parameters.

Returns: An enumerated return code indicating success (VE_RETURN_CODE_SUCCESS) or an error code. The returned enumeration type, **t_ve_return_code_e**, is described in Appendix B.72 on Page 510. Descriptions of all the available VeriCall Edge return/error codes is provided in Table A.1 on Page 245.

See Also: None

Description: **VE_NATStopKeepAlive** terminates a keep-alive operation for the specified network connection that was previously started using **VE_NATStartKeepAlive**. The function returns immediately and, if successful, the NAT traversal module will attempt to stop the keep-alive operation on the specified network connection. Once the keep-alive operation is stopped, one final stopped event will be reported via VE_USER_MSG_NAT_KEEP_ALIVE_EVENT event. Any failure to stop the keep-alive operation will be reported via a VE_USER_MSG_NAT_REQUEST_FAILED event.

The ve_application_id argument must be a valid registered application identifier.

The network connection structure is pointed to by *network_connection* and can be copied directly from a VE_USER_MSG_NETWORK_CONNECTION_EVENT indicating a closed connection (recommended), or it can be set manually. This structure is of type **t_ve_network_connection_s** and is described in Appendix B.64 on Page 498.

NAT binding keep-alive operations should only be attempted after a call has been made to VE_NATStartDiscover.

Continued >>

Example: This example shows how to stop a keep-alive operation:

```
#include <ve_nat_prototypes.h>

/* Stop keep-alive mechanism for terminated network connection
   reported by the VeriCall Edge system via a
   VE_USER_MSG_NETWORK_CONNECTION_EVENT event <event> */
{
   t_ve_return_code_e ret_code;
   t_ve_network_connection_s *network_connection =
                &event.message.network_connection_event.connection;

   if(event->message.network_connection_event.event ==
                                  VE_NETWORK_CONNECTION_EVENT_CLOSED)
   {
      ret_code = VE_NATStopKeepAlive( application_id,
                                      network_connection);

      if(ret_code != VE_RETURN_CODE_SUCCESS)
      {
          /* Handle error */
      }
   }
}
```

Appendix A

Return/Error Codes

This appendix details the return and error codes used throughout the VeriCall Edge API.

A.1 Return/Error Codes

Table A.1 below provides a complete list of return/error codes for the VeriCall Edge API functions (as defined in **ve_error.h**). Note that each VeriCall Edge API function description provides a list of possible return/error codes for the individual function.

Table A.1: VeriCall Edge API Return/Error Codes

Return/Error Code	Value	Description
VE_RETURN_CODE_ SUCCESS	0x0000	Function completed without error.
VE_RETURN_CODE_ ALREADY_INITIALIZED	0x0001	The VeriCall Edge system has already been initialized by the **VE_Initialize** function.
VE_RETURN_CODE_ UNINITIALIZED	0x0002	The VeriCall Edge system has not yet been initialized; use the **VE_Initialize** function to initialize the VeriCall Edge system.
VE_RETURN_CODE_ SIP_INITIALIZE_ERROR	0x0003	The SIP stack failed to initialize. The most likely cause of this error is that the SIP stack failed to obtain access to the default port number.
VE_RETURN_CODE_ INVALID_ALG_CONFIG	0x0004	The specified algorithm configuration structure is invalid.
VE_RETURN_CODE_ SCHEDULER_CONFIG_FAIL	0x0005	The specified scheduler configuration is invalid.
VE_RETURN_CODE_ QUEUE_ALLOCATION_ERROR	0x0006	The VeriCall Edge system failed to allocate resources for the internal message queue mechanism.
VE_RETURN_CODE_ CALLBACKS_NOT_REGISTERED	0x0007	At least one of the required VeriCall Edge Data Manager Interface callback functions, described in **Chapter 6** on Page 77, has not been registered by the user application.
VE_RETURN_CODE_ INSUFFICIENT_CONTEXT_SPACE	0x0008	There was insufficient context space defined to perform the tasks requested.

Return/Error Code	Value	Description
VE_RETURN_CODE_ ALREADY_COMMITTED	0x0009	The VeriCall Edge system has already received a valid **VE_Commit** function call.
VE_RETURN_CODE_ NOT_COMMITED	0x000A	The VeriCall Edge system configuration and system setup has not been committed with a valid **VE_Commit** function call.
VE_RETURN_CODE_ COMMIT_FAILED	0x000B	The **VE_Commit** function call failed.
VE_RETURN_CODE_ H323_INITIALIZE_ERROR	0x000C	The H.323 call control stack failed to initialize.
VE_RETURN_CODE_ NAT_INITIALIZE_ERROR	0x000D	The NAT traversal module failed to initialize correctly.
VE_RETURN_CODE_ ECAN_CONFIG_FAIL	0x0100	The ECAN configuration failed.
VE_RETURN_CODE_ AJBM_CONFIG_FAIL	0x0101	The AJBM configuration failed.
VE_RETURN_CODE_ PRE_POST_PROC_CONFIG_FAIL	0x0102	The configuration for the VeriCall Edge pre/post processor configuration failed.
VE_RETURN_CODE_ SIP_CONFIG_FAIL	0x0105	The SIP configuration failed.
VE_RETURN_CODE_ NTG_CONFIG_FAIL	0x0106	The Named Tone Generator configuration failed.
VE_RETURN_CODE_ NTD_CONFIG_FAIL	0x0107	The Named Tone Detector configuration failed.
VE_RETURN_CODE_ CN_CONFIG_FAIL	0x0108	The Comfort Noise configuration failed.
VE_RETURN_CODE_ G711U_CONFIG_FAIL	0x0109	The G.711 μ-Law configuration failed.
VE_RETURN_CODE_ G711A_CONFIG_FAIL	0x010A	The G.711 a-Law configuration failed.
VE_RETURN_CODE_ G729_CONFIG_FAIL	0x010B	The G.729 configuration failed.
VE_RETURN_CODE_ TE_CONFIG_FAIL	0x010C	The Telephony Event configuration failed.
VE_RETURN_CODE_ DEFAULT_CHANNEL_CONFIG_FAIL	0x010D	The default channel configuration failed.
VE_RETURN_CODE_ IP_CONFIG_FAIL	0x010E	The VeriCall Edge system IP configuration failed.
VE_RETURN_CODE_ H263_CONFIG_FAIL	0x010F	The H.263 video configuration failed.
VE_RETURN_CODE_ GSM_AMR_CONFIG_FAIL	0x0110	The GSM-AMR configuration failed.
VE_RETURN_CODE_ DATA_END_POINT_CONFIG_FAIL	0x0111	The configuration for the data-end point was invalid and failed.
VE_RETURN_CODE_ H323_CONFIG_FAIL	0x0112	The H.323 call control stack configuration failed.

Return/Error Code	Value	Description
VE_RETURN_CODE_ TX_LIP_SYNC_CONFIG_FAIL	0x0113	The transmit lip-synchronization configuration failed.
VE_RETURN_CODE_ G723_CONFIG_FAIL	0x0114	The G.723.1 configuration failed.
VE_RETURN_CODE_ LOGGING_CONFIG_FAIL	0x0115	The logging configuration failed.
VE_RETURN_CODE_ STATISTICS_CONFIG_FAIL	0x0116	The statistics configuration failed.
VE_RETURN_CODE_ TEAR_PROTECT_CONFIG_FAIL	0x0117	The configuration for video tear protection failed.
VE_RETURN_CODE_ NAT_CONFIG_FAIL	0x0118	The NAT traversal module configuration failed.
VE_RETURN_CODE_ NAT_NETWORK_ERROR	0x0119	The NAT traversal module detected a network error on the transport layer.
VE_RETURN_CODE_ NAT_NETWORK_TIMEOUT	0x0120	The NAT traversal module timed out waiting for a response to a request.
VE_RETURN_CODE_ NAT_CANCELLED	0x0121	The NAT **VE_NATCancelDiscover** function has been used before the **VE_NATStartDiscover** function completed.
VE_RETURN_CODE_ MPEG4_CONFIG_FAIL	0x0122	The MPEG4 configuration failed.
VE_RETURN_CODE_ G726_CONFIG_FAIL	0x0123	The G.726 configuration failed.
VE_RETURN_CODE_ LAST_STREAM_INITIALIZATION_FAILED	0x0124	The last stream configuration failed. The last stream configuration structure is **t_ve_last_stream_configuration_s** as described in Appendix 257 on Page B.5.
VE_RETURN_CODE_ DEFAULT_STREAM_STRUCTURE_FAILED	0x0125	The default stream configuration failed. The last stream configuration structure is **t_ve_stream_configuration_s** as described in Appendix 298 on Page B.24.
VE_RETURN_CODE_ILBC_CONFIG_FAIL	0x0126	The iLBC configuration failed.
VE_RETURN_CODE_GAIN_CONTROL_ CONFIG_FAIL	0x0127	The Gain Control configuration failed.
VE_RETURN_CODE_ NULL_POINTER	0x0200	At least one pointer argument passed to the function is `NULL`.
VE_RETURN_CODE_ INVALID_HANDLE	0x0201	The handle passed to the function is invalid for the function operation.
VE_RETURN_CODE_ INVALID_APPLICATION	0x0202	The application ID passed to the function has not been registered with the VeriCall Edge system. One of the first function calls made in the user application should be to **VE_RegisterApp**.
VE_RETURN_CODE_ INVALID_USER	0x0203	The specified user is invalid, possibly the user has not been registered with the VeriCall Edge system via **VE_AddUser**.
VE_RETURN_CODE_ INVALID_ATTRIBUTE_INDEX	0x0204	The attribute index used was invalid. Table C.2 on Page 662 gives the associated attribute index for each available attribute type.

Return/Error Code	Value	Description
VE_RETURN_CODE_ TOO_MANY_DIGITS_IN_STRING	0x0205	The string length exceeds 30 digits in a call to **VE_GenerateDigitString**.
VE_RETURN_CODE_ TONE_NOT_SUPPORTED	0x0206	The VeriCall Edge system does not support the specified tone. If the required tone is not available through **VE_GenerateNamedTone**, a user defined tone can be used via **VE_GenerateUserTone**.
VE_RETURN_CODE_ STATS_TYPE_UNKNOWN	0x0207	The statistics type passed as an argument to **VE_GetStatistics** is unknown.
VE_RETURN_CODE_ STATS_TYPE_CANNOT_BE_RESET	0x0208	The statistics type passed as an argument to **VE_ClearStatistics** cannot be reset.
VE_RETURN_CODE_ INVALID_AUDIO_CONTROL_TYPE	0x0209	The audio control type argument to **VE_ControlAudio** is invalid.
VE_RETURN_CODE_ INVALID_AUDIO_CONTROL_DIRECTION	0x020A	The audio control direction type argument to **VE_ControlAudio** is invalid.
VE_RETURN_CODE_ INVALID_CHANNEL_CONFIGURATION	0x020B	The specified channel is not valid or has not yet been configured.
VE_RETURN_CODE_ INVALID_AUDIO_FORMAT	0x020C	The VeriCall Edge system does not support the audio format.
VE_RETURN_CODE_ STORE_ENTRY_READ_ONLY	0x020D	An attribute of The VeriCall Edge system cannot be written to because it is read-only.
VE_RETURN_CODE_ USER_ALREADY_KNOWN	0x020E	The specified user has already been registered with the VeriCall Edge system.
VE_RETURN_CODE_INVALID_ VIDEO_CONTROL_TYPE	0x020F	An invalid video control type was requested in a call to **VE_ControlVideo**.
VE_RETURN_CODE_INVALID_ VIDEO_CONTROL_FILTER_SELECTION	0x0210	An invalid video filter selection has been made. Valid settings are those given in **t_ve_video_control_filter_selector_e** as described in Appendix B.150 on Page 576.
VE_RETURN_CODE_INVALID_ VIDEO_CONTROL_DISPLAY_STATE	0x0211	An invalid video display state has been selected. Valid settings are those given in **t_ve_video_control_select_display_state_e** as described in Appendix B.151 on Page 576.
VE_RETURN_CODE_INVALID_ VIDEO_CONTROL_SCALE_FACTOR	0x0212	An invalid video scale factor has been set.
VE_RETURN_CODE_INVALID_ VIDEO_CONTROL_ROTATION_FACTOR	0x0213	An invalid video rotation angle has been set. Valid settings are those given in **t_ve_video_control_rotate_e** as described in Appendix B.152 on Page 577.
VE_RETURN_CODE_INVALID_ VIDEO_CONTROL_MIRROR_STATE	0x0214	An invalid mirror plane has been set for the video image. Valid settings are those given in **t_ve_video_control_mirror_e** as described in Appendix B.153 on Page 577.
VE_RETURN_CODE_INVALID_ VIDEO_CONTROL_MUTE_MODE	0x0215	An invalid video mute mode has been selected. Valid settings are those given in **t_ve_video_control_mute_mode_e** as described in Appendix B.154 on Page 578.

Return/Error Code	Value	Description
VE_RETURN_CODE_INVALID_ VIDEO_CONTROL_ENCODE_FORMAT	0x0216	The VeriCall Edge system does not support the video encode format. Valid settings are those given in **t_ve_video_control_encode_format_e** as described in Appendix B.156 on Page 579.
VE_RETURN_CODE_INVALID_ VIDEO_CONTROL_DATA_POINT_CONFIG	0x0217	An unavailable data-end point has been selected.
VE_RETURN_CODE_INVALID_ VIDEO_CONTROL_IMAGE_FORMAT	0x0218	An invalid video image format has been selected. Valid settings are those given in **t_ve_video_control_encode_format_e** as described in Appendix B.156 on Page 579.
VE_RETURN_CODE_INVALID_VIDEO_ CONTROL_LIP_SYNC_ADJUSTMENT	0x0219	An invalid lip-synchronization adjustment level has been requested.
VE_RETURN_CODE_INVALID_VIDEO_ CONTROL_RESET_CONFIG_TYPE	0x021A	The alteration of either the video pre-processor or video post-processor is incorrect and could not be completed.
VE_RETURN_CODE_INVALID_ CAPABILITY_MISMATCH	0x021B	The channel cannot perform the requested action as it does not have that capability.
VE_RETURN_CODE_ ATTRIBUTE_OUT_OF_RANGE_ERROR	0x021C	At least one of the referenced enumerated types is out of the specified ranges of the enumeration.
VE_RETURN_CODE_ MEDIA_EXCEEDS_PACKET_BUDGET	0x021D	The packet size specified is greater than the codecs configured value.
VE_RETURN_CODE_CODEC_CONFIG_ INCOMPATIBLE_PRE_PROCESSOR	0x021E	The specified code configuration is incompatible with the pre-processor.
VE_RETURN_CODE_ILLEGAL_ SCALING_FACTOR	0x021F	The specified scaling factor is not a valid value.
VE_RETURN_CODE_ INVALID_ATTRIBUTE_SIZE	0x0220	The supplied `size` argument in an AP{I function used to access (read/write) the VeriCall Edge attribute store, does not match that expected for the specified attribute type.
VE_RETURN_CODE_ STORE_ENTRY_WRITE_ONLY	0x0221	An attribute type is write-only, but **VE_GetSystemAttribute** or **VE_RemoveSystemAttribute** has been used to try and access the attribute type. This occurs when the global IP settings have been changed, via **VE_UpdateSystemAttribute**, making the global IP settings invalid; therefore they cannot be read.
VE_RETURN_CODE_ CHANNEL_OUT_OF_RANGE	0x0222	The specified channel is not within the VeriCall Edge system range. This range was only set through the **t_ve_init_config_s** structure, described in Appendix B.2 on Page 254, used in the **VE_Initialize** function.
VE_RETURN_CODE_INVALID_VIDEO CONTROL_INVALID_DATA_END_POINT	0x0230	An invalid data-end point has been requested.
VE_RETURN_CODE_INVALID_ NETWORK_TOS_CONFIGURATION	0x0240	An invalid Type of Service (ToS) value has been selected. Valid settings are given in **t_ve_ip_settings_default_configuration_s** as described in Appendix B.8 on Page 261.

Return/Error Code	Value	Description
VE_RETURN_CODE_INVALID_ TRANSMISSION_MEDIA_TYPE	0x0251	An invalid media transmission type has been requested.
VE_RETURN_CODE_ BEEP_SELECTION_NOT_SUPPORTED	0x0252	Reserved for future use.
VE_RETURN_CODE_SIP_ADD_ USER_REGISTRATION_DATA_FAILED	0x0260	The VeriCall Edge system was unable to add the SIP user due to invalid registration data. This error is the result of a call to **VE_SIPAddUsersRegistrationData**.
VE_RETURN_CODE_SIP_GET_ USER_REGISTRATION_DATA_FAILED	0x0261	Registering the user data failed. This error is the result from a call to **VE_SIPGetUsersRegistrationData**.
VE_RETURN_CODE_SIP_UPDATE_ USER_REGISTRATION_DATA_FAILED	0x0262	The VeriCall Edge system was unable to update the SIP user due to invalid registration data. This error is the result of a call to **VE_SIPUpdateUsersRegistrationData**
VE_RETURN_CODE_SIP_REMOVE_ USER_REGISTRATION_DATA_FAILED	0x0263	The VeriCall Edge system was unable to remove the SIP user due to invalid registration data. This error is the result of a call to **VE_SIPRemoveUsersRegistrationData**
VE_RETURN_CODE_SIP_ADD_ USER_PRESENCE_DATA_FAILED	0x0264	The VeriCall Edge system was unable to add the SIP user presence data. This error is the result of a call to **VE_SIPAddUsersPresenceData**.
VE_RETURN_CODE_SIP_UPDATE_ USER_PRESENCE_DATA_FAILED	0x0265	The VeriCall Edge system was unable to update the SIP user presence data. This error is the result of a call to **VE_SIPUpdateUsersPresenceData**.
VE_RETURN_CODE_SIP_REMOVE_ USER_PRESENCE_DATA_FAILED	0x0266	The VeriCall Edge system was unable to remove the SIP user presence data. This error is the result of a call to **VE_SIPRemoveUsersPresenceData**.
VE_RETURN_CODE_SIP_GET_ USER_PRESENCE_DATA_FAILED	0x0267	The VeriCall Edge system was unable to retrieve the SIP user presence data. This error is the result of a call to **VE_SIPGetUsersPresenceData**.
VE_RETURN_CODE_SIP_INVALID_ CALL_HOLD_METHOD	0x0268	The selected call hold method is invalid. Valid values are those given in **t_ve_sip_call_hold_method_e** described in Appendix B.190 on Page 609.
VE_RETURN_CODE_SIP_INVALID_ MW_START_FLAG	0x0269	The selected Message Waiting flag is invalid. Valid values are those given in **t_ve_sip_mw_start_flag_e** described in Appendix B.196 on Page 616.
VE_RETURN_CODE_H323_ADD_ USER_REGISTRATION_DATA_FAILED	0x0270	The VeriCall Edge system was unable to add the H.323 user due to invalid registration data. This error is the result of a call to **VE_H323AddUsersRegistrationData**.
VE_RETURN_CODE_H323_UPDATE_ USER_REGISTRATION_DATA_FAILED	0x0271	The VeriCall Edge system was unable to update the H.323 user due to invalid registration data. This error is the result of a call to **VE_H323UpdateUsersRegistrationData**.

Return/Error Code	Value	Description
VE_RETURN_CODE_H323_REMOVE_ USER_REGISTRATION_DATA_FAILED	0x0272	The VeriCall Edge system was unable to remove the H.323 user due to invalid registration data. This error is the result of a call to **VE_H323RemoveUsersRegistrationData**.
VE_RETURN_CODE_ MEMORY_ALLOC_FAIL	0x0300	An attempt to assign memory has failed.
VE_RETURN_CODE_ MEMORY_DECALLOC_FAIL	0x0301	An attempt to de-allocate a previously allocated area of memory has failed.
VE_RETURN_CODE_ USER_ALLOC_TOO_SMALL	0x0302	The memory allocated to a function argument is too small to hold the necessary information.
VE_RETURN_CODE_ CORRUPT_POINTER	0x0303	At least one pointer argument passed to the function is invalid.
VE_RETURN_CODE_ NO_MESSAGES	0x0400	The request to access a message in the queue is invalid as there is no message available.
VE_RETURN_CODE_ FAILED_TO_SEND_MESSAGE	0x0401	A message failed to be added to the queue for an unspecified reason.
VE_RETURN_CODE_ MESSAGE_TOO_BIG	0x0402	An internal message queue error occurred.
VE_RETURN_CODE_ RESOURCE_QUERY_FAIL	0x0500	The **VE_MemoryResourceQuery** function call failed.
VE_RETURN_CODE_ API_NOT_SUPPORTED_ON_PLATFORM	0x0501	The VeriCall Edge API function is not supported on the current platform.
VE_RETURN_CODE_FUNCTIONALITY_ ALREADY_ACTIVATED	0x0502	The VeriCall Edge software functionality is already active.
VE_RETURN_CODE_FUNCTIONALITY_ ALREADY_DEACTIVATED	0x0503	The VeriCall Edge software functionality is already de-active.
VE_RETURN_CODE_ ILLEGAL_PARAMETER	0x0504	At least one of the supplied configuration parameters is invalid.
VE_RETURN_CODE_INVALID_STATE	0x0505	Reserved for future use.
VE_RETURN_CODE_ NO_STORE_ENTRY	0x0600	A requested store entry is not valid, either it is `NULL` or has been corrupted.
VE_RETURN_CODE_ CHANNEL_NOT_AVAILABLE	0x0601	The specified channel data entity cannot be found.
VE_RETURN_CODE_ MEDIA_PROFILE_NOT_FOUND	0x0602	The requested media profile data entity has not been set or has been deleted.
VE_RETURN_CODE_ LINE_ECAN_ID_NOT_FOUND	0x0603	The specified VeriCall Edge line ECAN identifier is not valid. Valid settings are given in **t_ve_line_ecan_id_e** as described in Appendix B.89 on Page 526.
VE_RETURN_CODE_ ACOUSTIC_ECAN_ID_NOT_FOUND	0x0604	The specified VeriCall Edge acoustic ECAN identifier is not valid. Valid settings are given in **t_ve_acoustic_ecan_id_e** as described in Appendix B.87 on Page 524.
VE_RETURN_CODE_ DTMF_CONSTRUCTION_FAIL	0x0700	The DTMF series play length exceeded `VC_TONE_MAX_SERIES_COUNT` as defined in **ve_tones.h**.

Return/Error Code	Value	Description
VE_RETURN_CODE_ INTERNAL_ERROR	0x0701	The VeriCall Edge system has reported an internal error.
VE_RETURN_CODE_CHANNEL_ VAD_AND_CNG_CONFIG_FAILED	0x0800	The VAD and/or CNG configuration failed.
VE_RETURN_CODE_ START_STREAM_FAILED	0x0801	An attempt to start a media stream failed. This error will result from a failed call to one of the following: **VE_SIPSessionStart** **VE_SIPSessionStart_WithConfig** **VE_H323StartCall**
VE_RETURN_CODE_ START_STREAM_INACTIVE	0x0802	An action has been requested on an inactive media stream. Normally a result of an attempt to use, for example, **VE_SIPSessionUpdate** after a call to **VE_SIPSessionStop** has been made.
VE_RETURN_CODE_ START_STREAM_DUPLICATED	0x0803	An action has been requested that has already been applied to a media stream. Normally this is the result of, for example, **VE_SIPSessionStart** called after a previous **VE_SIPSessionStart** call.
VE_RETURN_CODE_ STOP_STREAM_FAILED	0x0804	The request to stop a media stream failed for an unspecified reason. This may result from a call to **VE_SIPSessionStop** or **VE_H323StopCall**.
VE_RETURN_CODE_ TONE_GENERATE_FAILED	0x0805	An attempt to generate a tone has failed. This may be returned from a **VE_GenerateNamedTone** or **VE_GenerateUserTone** function call.
VE_RETURN_CODE_ PLAY_PROMPT_FAILED	0x0806	An attempt to play a file into the media stream failed.
VE_RETURN_CODE_ RECORD_PROMPT_FAILED	0x0807	An attempt to record the media stream failed.
VE_RETURN_CODE_ FATAL_ERROR_CHANNEL_CREATE	0x0900	There was a fatal error in creating a channel.
VE_RETURN_CODE_ FATAL_ERROR_ILLEGAL_AJBM_CONFIG	0x0901	There was a fatal error in trying to configure the AJBM. The AJBM is configured through the **t_ve_ajbm_algorithm_default_configuration_s** structure, described in Appendix B.9 on Page 262.

Appendix B

Structures and Enumerations

This Appendix details the structures and enumerations used within the VeriCall Edge API functions.

VeriCall Edge Structures

B.1 t_ve_system_s

The **t_ve_system_s** structure is used in calls to the **VE_RunVeriCallEdge** function. The structure is written by the VeriCall Edge system to inform the user application of the number of queued messages in the queue. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  UWord32    message_in_queue;
}t_ve_system_s;
```

The element of this structure is as follows:

Element	Definition
message_in_queue	The number of messages in the queue for this registered user application.

B.2 t_ve_init_config_s

The **t_ve_init_config_s** structure is used in calls to the **VE_Initialize** function. The structure is used for the VeriCall Edge system initialization parameters. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  UWord32    number_of_channels;
  UWord32    number_of_users;
  UWord32    number_of_interfaces;
}t_ve_init_config_s;
```

The elements of this structure are as follows:

Element	Definition
number_of_channels	The maximum number of channels that user applications will add to the VeriCall Edge system.
number_of_users	The maximum number of users that user applications will add to the VeriCall Edge system.
number_of_interfaces	The total number of Data Manager Interfaces (callbacks) that the user application will register with the VeriCall Edge system.

The number of callbacks to be registered with the VeriCall Edge system will depend on the number of channels to be created and the whether both audio and video channels are to be created. Please refer the VeriCall Edge User Manual for more information.

B.3 t_ve_version_s

The **t_ve_version_s** structure is used in calls to the **VE_GetVersion** function to store the VeriCall Edge software version information. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  UWord32    major;
  UWord32    minor;
  Uword32    patch;
  Word8      qualifier[VE_MAX_QUALIFIER_SIZE];
  Word8      time[VE_MAX_TIME_SIZE];
  Word8      date[VE_MAX_DATE_SIZE];
}t_ve_version_s;
```

The elements of the **t_ve_version_s** structure are as follows:

Element	Definition
major	Major version number.
minor	Minor version number.
patch	Patch version number.
qualifier	An optional NULL terminated string qualifier of the version, which cannot exceed VE_MAX_QUALIFIER_SIZE characters in length (currently defined as being 50 characters including the termination character).
time	An optional NULL terminated string representation of the time of the build, which cannot exceed VE_MAX_TIME_SIZE characters in length (currently defined as being 50 characters including the termination character).
date	An optional NULL terminated string representation of the date of the build, which cannot exceed VE_MAX_DATE_SIZE characters in length (currently defined as being 50 characters including the termination character).

For example, a version number of 2.1.0 would provide major version number 2, minor version number 1 and patch version number 0.

B.4 t_ve_system_logging_s

The **t_ve_system_logging_s** structure is used to configure the level of logging to take place within the VeriCall Edge system. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_verbose_level_e                  verbose;
  t_ve_system_logging_method_e          method;
  t_ve_system_logging_destination_e     destination;
  UWord32                               file_handle;
}t_ve_system_logging_s;
```

The element of this structure is as follows:

Element	Definition
verbose	Enumerated verbose level of system logging. The enumerated type, **t_ve_verbose_level_e**, is defined in Appendix B.73 on Page 513.
method	An enumeration of type **t_ve_system_logging_method_e** as described in Appendix B.75 on Page 515. This enumerated type allows the definition of the method of the system logging.
destination	An enumeration of type **t_ve_system_logging_destination_e** as described in Appendix B.74 on Page 514. This enumerated type allows the location of the log file to defined.
file_handle	The file handle used for log *destination* element VE_SYSTEM_LOGGING_METHOD_FILEHANDLE. This filehandle can be used for stdout, stderr or a file and allows increased flexibility to the user application.

The VeriCall Edge system specific debug output is for use by Trinity Convergence and should not be interpreted by the user.

B.5 t_ve_last_stream_configuration_s

The **t_ve_last_stream_configuration_s** attribute structure can be retrieved using the **VE_GetSystemAttribute** function. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The **t_ve_last_stream_configuration_s** structure is populated and stored when a **VE_StartStream** function is successfully used. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_last_stream_state_e        state;
  t_ve_stream_configuration_s     stream;
}t_ve_last_stream_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
state	An enumerated type, **t_ve_last_stream_state_e** as described in Appendix B.131 on Page 558, that returns the current state of the stream configuration structure.
stream	A structure of type **t_ve_stream_configuration_s**, as described in Appendix B.24 on Page 298. This structure will contain the stored stream configuration.

B.6 t_ve_run_vericall_edge_config_s

The **t_ve_run_vericall_edge_config_s** attribute structure is used within calls to **VE_RunVeriCallEdgeWithConfig** function. The structure allows the configuration of the VeriCall Edge classes and modules and their state. This can be used to control resource usage within the VeriCall Edge system. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_module_class_e          class_select;
  t_ve_module_state_class_u    module_state_class;
}t_ve_run_vericall_edge_config_s;

typedef union{
  t_ve_audio_video_module_state_s      audio_video_module_state;
  t_ve_audio_module_state_s            audio_only_module_state;
}t_ve_module_state_class_u;
```

The elements of this structure are as follows:

Element	Definition
class_select	An enumerated type, **t_ve_module_class_e** as described in Appendix B.133 on Page 559, that allows the selection of the modules to which the operation is to be applied.
module_state_class	A union of structures whose selection is determined by the *class_select* enumerated type. The selected structures can be one of the following: **t_ve_audio_module_state_s** as described in Appendix B.6.2 on Page 259. **t_ve_audio_video_module_state_s** as described in Appendix B.6.1 on Page 259.

A class within VeriCall Edge is a device that is capable of doing audio, video or both. A module with VeriCall Edge is the configuration associated with a class.

B.6.1 t_ve_audio_video_module_state_s

The **t_ve_audio_video_module_state_s** structure is used by the **t_ve_run_vericall_edge_config_s** structure defined in Appendix B.6 on Page 258. The structure allows the low-power mode state of the sub-modules of this module to be set. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_module_state_e        audio;
  t_ve_module_state_e        video_pre_processor;
  t_ve_module_state_e        video_encoder;
  t_ve_module_state_e        video_post_processor;
  t_ve_module_state_e        video_decoder;
}t_ve_audio_video_module_state_s;
```

The elements of this structure are as follows:

Element	Definition
audio	An enumerated type, **t_ve_module_state_e** as described in Appendix B.132 on Page 559. the type allows the low-power mode state of the audio module to be set.
video_pre_processor	An enumerated type, **t_ve_module_state_e** as described in Appendix B.132 on Page 559. the type allows the low-power mode state of the video pre-processor module to be set.
video_encoder	An enumerated type, **t_ve_module_state_e** as described in Appendix B.132 on Page 559. the type allows the low-power mode state of the video encoder module to be set.
video_post_processor	An enumerated type, **t_ve_module_state_e** as described in Appendix B.132 on Page 559. the type allows the low-power mode state of the video post-processor module to be set.
video_decoder	An enumerated type, **t_ve_module_state_e** as described in Appendix B.132 on Page 559. the type allows the low-power mode state of the video decoder module to be set.

B.6.2 t_ve_audio_module_state_s

The **t_ve_audio_module_state_s** structure is used by the **t_ve_run_vericall_edge_config_s** structure defined in Appendix B.6 on Page 258. The structure allows the low-power mode state of the sub-modules of this module to be set. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_module_state_e        audio;
}t_ve_audio_module_state_s;
```

The elements of this structure are as follows:

Element	Definition
audio	An enumerated type, **t_ve_module_state_e** as described in Appendix B.132 on Page 559. the type allows the low-power mode state of the audio module to be set.

B.7 t_ve_ip_default_configuration_s

The **t_ve_ip_default_configuration_s** structure is used for the VeriCall Edge system configuration and Table C.1 on Page 659 shows the VeriCall Edge attribute associated with this structure. The structure is used to define the IP interface of the VeriCall Edge system. This structure is defined in **ve_configuration.h** and is described below:

```
typedef struct{
  t_ve_ip_default_configuration_e   ip_configuration;
  t_ve_ip_default_address_u         ip_address;
  t_ve_ip_port_reopen_e             ip_port_reopen;
  t_ve_ip_default_attributes_s      ip_attributes;
}t_ve_ip_default_configuration_s;

typedef union{
 Word8 ip_domain[VE_IP_CONFIGURATION_MAX_FQDN_ADDRESS_STRING_SIZE];
 UWord32  ip_address;
}t_ve_ip_default_address_u;
```

The elements of this structure are as follows:

Element	Definition
ip_configuration	Enumerated type of **t_ve_ip_default_configuration_e**, as defined in Appendix B.81 on Page 520, to determine the IP configuration type to be used.
ip_address	A union of elements that can be one of the following: **ip_domain** A fully qualified domain name which cannot exceed `VE_IP_CONFIGURATION_MAX_FQDN_ADDRESS_STRING_SIZE` characters in length (currently defined to be 100 characters). **ip_address** An IP address represented in network byte order hexadecimal.
ip_port_reopen	An enumerated type, **t_ve_ip_port_reopen_e** as described in Appendix B.82 on Page 520, that is a flag allowing this particular functionality to be enabled/disabled. If the VeriCall Edge IP address is changed, this functionality when enabled allows the VeriCall Edge system to automatically close the previous IP port and open the new port number. If this functionality is disable the previous port number remains open.
ip_attributes	A structure, **t_ve_ip_default_attributes_s**, as defined in Appendix B.7.1 on Page 261. The elements of this structure allowing IP configuration parameters to be set.

The VeriCall Edge system uses the network byte ordering for IP addresses when a UWord32 value is required.

B.7.1 t_ve_ip_default_attributes_s

The **t_ve_ip_default_attributes_s** structure is used by the **t_ve_ip_default_configuration_s** structure defined in Appendix B.7 on Page 260. The **t_ve_ip_default_attributes_s** structure allows specific IP attributes to be defined for the VeriCall Edge IP interface. This structure is defined in **ve_configuration.h** and is described below:

```
typedef struct{
  UWord32    ve_max_ip_rate_kbits_per_sec;
  UWord32    ve_max_ip_packet_size;
}t_ve_ip_default_attributes_s;
```

The elements of this structure are as follows:

Element	Definition
ve_max_ip_rate_kbits_per_sec	Defines the maximum packet rate for the IP network that the audio and video packet stream must fit within. In this context, 'k' is defined as being 1000 bits.
ve_max_ip_packet_size	The maximum packet size that can be received and transmitted.

B.8 t_ve_ip_settings_default_configuration_s

The **t_ve_ip_settings_default_configuration_s** structure is used for the VeriCall Edge system configuration and Table C.1 on Page 659 shows the VeriCall Edge attribute associated with this structure. The structure is used to define the configuration of the IP interface of the VeriCall Edge system. This is known as the VeriCall Edge global IP settings, individual VeriCall Edge module IP settings can be configured separately using this structure as well. This structure is defined in **ve_configuration.h** and is described below:

```
typedef struct{
  UWord8       ve_tos;
}t_ve_ip_settings_default_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
ip_configuration	The Type Of Service (TOS) to be used. The TOS can be as per the IP TOS (RFC0791 and RFC1349) or Differentiated Services TOS (RFC2474).

If ANY of the module default IP settings are changed with the VE_SetSystemAttribute API function or the VE_UpdateSystemAttribute API function, then the global IP settings attribute type, `VE_ATTRIBUTE_TYPE_DEFAULT_IP_SETTINGS` can no longer be accessed.

B.9 t_ve_ajbm_algorithm_default_configuration_s

The **t_ve_ajbm_algorithm_default_configuration_s** structure contains default configuration parameters for the VeriCall Edge Adaptive Jitter Buffer Manager (AJBM) task. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  t_ve_default_configuration_boolean_e           ajbm_enable;
  t_ve_default_configuration_boolean_e
                                    ajbm_dynamic_adjustment_state;
  Word32                                         minimumbuffer_ms;
  Word32                                         maximumbuffer_ms;
  Word32                                         initialbuffer_ms;
  t_ve_ajbm_algorithm_default_lip_sync_state_e   lip_sync_state;
  t_ve_ajbm_algorithm_default_lip_sync_config_s  lip_sync_config;
}t_ve_ajbm_algorithm_default_configuration_s;
```

The elements of the **t_ve_ajbm_algorithm_default_configuration_s** structure are described below:

Element	Definition
ajbm_enable	Enumerated type to enable/disable adaptive jitter buffer management. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521.
ajbm_dynamic_adjustment_state	An enumerated type, **t_ve_default_configuration_boolean_e** as described in Appendix B.84 on Page 521, that is a flag to enable/disable the jitter buffers ability to dynamically adapt. When disabled, the jitter buffer will stop adapting and will allows data modems to work with the VeriCall Edge system.
minimumbuffer_ms	The minimum size of the adaptive jitter buffer (in milliseconds).
maximumbuffer_ms	The maximum size of the adaptive jitter buffer (in milliseconds).
initialbuffer_ms	The initial size of the adaptive jitter buffer (in milliseconds).
lip_sync_state	Enables or disables the lip-synchronization facility of this AJBM. The ability to enable or disable this functionality is contained within the enumerated type **t_ve_ajbm_algorithm_default_lip_sync_state_e**, defined in Appendix B.90 on Page 527.
lip_sync_config	Defines configuration parameters for an enabled lip-synchronization capability on this AJBM. The configuration parameters are contained within the **t_ve_ajbm_algorithm_default_lip_sync_config_s** structure, defined in Appendix B.9.1 on Page 263.

In order to get the most out of the VeriCall Edge system, the AJBM buffer elements (configurable on a per channel basis) should be set to match the particular network characteristics.

Continued >>

For a low impairment network, the VeriCall Edge system should be set to add minimal delay. For audio channels this can be achieved by configuring the initial buffer size to 20ms, the minimum size should be 0ms and the maximum size should be 60ms. For a high impairment network, the VeriCall Edge system should be set to add more delay to absorb such impairment cleanly. You can achieve this by configuring the initial buffer size to 60ms, the minimum size should be 0ms and the maximum size should be 200ms.

AJBM settings for a video channel should have the initial buffer size set to 0ms, the minimum size should be 0ms and the maximum size should be 200ms

Maximum buffer depth is constrained by the configuration word size, internal AJBM configuration size and the type of algorithm used. For example, the maximum buffer depth for G.711 is 3500ms while for video this is 300ms (because of the increased packet size). The principle disadvantage of increasing the AJBM buffer depth is an increase in memory usage.

B.9.1 t_ve_ajbm_algorithm_default_lip_sync_config_s

The **t_ve_ajbm_algorithm_default_lip_sync_config_s** structure is used within the **t_ve_ajbm_algorithm_default_configuration_s** structure, as defined in Appendix B.9 on Page. 262. The **t_ve_ajbm_algorithm_default_lip_sync_config_s** structure contains default parameters relating to the lip-synchronization configuration of the audio and video streams. This structure is defined in **ve_configuration.h** and is described below.

```
typedef struct{
  Word32     tolerance_in_msecs;
  Word32     max_manual_msec_adjustment;
}t_ve_ajbm_algorithm_default_lip_sync_config_s;
```

The elements of this structure are as follows:

Element	Definition
tolerance_in_msecs	The level of lip-synchronization “slip” that will be tolerated before lip-synchronization adjustment will begin.
max_manual_msec_adjustment	The maximum number of milliseconds that the lip-synchronization can be adjusted by.

B.10 t_ve_line_ecan_configuration_s

The **t_ve_line_ecan_configuration_s** structure contains default configuration parameters for the VeriCall Edge line Echo Canceller (ECAN) algorithm task. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  t_ve_line_ecan_id_e                ve_line_ecan_id;
  t_ve_line_ecan_configuration_u     ve_line_ecan_config;
}t_ve_line_ecan_configuration_s;

typedef union{
  t_ve_line_ecan_16ms_ecan_configuration_s     vc_16ms_ecan;
}t_ve_line_ecan_configuration_u;
```

The elements of this structure are described below:

Element	Definition
ve_line_ecan_id	An enumerated type used to define the type of line ECAN to be used. The enumerated type, **t_ve_line_ecan_id_e**, is defined in Appendix B.89 on Page 526.
ve_line_ecan_config	Union of structures used to define the configuration structure to be used for this line ECAN. The available structures, selected by the *ve_line_ecan_id* enumeration, is as follows: t_ve_line_ecan_16ms_ecan_configuration_s as defined in Appendix B.10.1.1 on Page 265.

There can only be one active VeriCall Edge Echo Canceller per-channel. If the Acoustic Echo Canceller (defined in Appendix B.11 on Page 266) is also enabled, this Line Echo Canceller will be the only active one within the channel, i.e. this Line Echo Canceller takes preference over the Acoustic Echo Canceller.

B.10.1.1 t_ve_line_ecan_16ms_ecan_configuration_s

The **t_ve_line_ecan_16ms_ecan_configuration_s** structure contains default configuration parameters for the VeriCall Edge 16ms line Echo Canceller (ECAN) algorithm task. This structure is part of the **t_ve_line_ecan_configuration_s** structure, described in Appendix B.10 on Page 264. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  t_ve_default_configuration_boolean_e          ecan_enable;
  t_ve_line_16ms_ecan_nlp_default_settings_e    nlp_settings;
  Word32                                        bulk_delay;
  t_ve_default_configuration_boolean_e        bulk_delay_estimator;
  Word32                                        search_range;
  t_ve_default_configuration_boolean_e          write_audio;
}t_ve_line_ecan_16ms_ecan_configuration_s;
```

The elements of this structure are described below:

Element	Definition
ecan_enable	Enumerated type to enable/disable the ECAN. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521. The ECAN should be turned off for high-speed (data) calls.
nlp_settings	Enumerated type to define the form of NLP to use. The enumerated type, **t_ve_line_16ms_ecan_nlp_default_settings_e**, is defined in Appendix B.86 on Page 523.
bulk_delay	When the bulk delay estimator is disabled then this setting defines the bulk delay. When the bulk delay estimator is enabled then this setting defines the start of the search range (see below). The bulk delay must be greater than zero and the default value is 50 (milliseconds).
bulk_delay_estimator	Enumerated type to enable/disable the bulk delay estimator. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521.
search_range	The length of the search range for the bulk delay estimator. The start of the search range is defined by the bulk delay setting (see above). Default value is 50 (milliseconds).
write_audio	Enumerated type used to write the audio inputs and outputs out for debugging purposes. The enumerated type, **t_ve_default_configuration_boolean_e** is described in Appendix B.84 on Page 521. By default this enumerated type is false. This element should only be used in consultation with Trinity Convergence.

B.11 t_ve_acoustic_ecan_configuration_s

The **t_ve_acoustic_ecan_configuration_s** structure contains default configuration parameters for the VeriCall Edge acoustic Echo Canceller (ECAN) algorithm task. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  t_ve_acoustic_ecan_id_e              ve_acoustic_ecan_id;
  t_ve_acoustic_ecan_configuration_u   ve_acoustic_ecan_config;
}t_ve_acoustic_ecan_configuration_s;

typedef union{
  t_ve_acoustic_ecan_aec_configuration_s          aec;
  t_ve_acoustic_ecan_aec_complex_configuration_s  aec_complex;
  t_ve_acoustic_ecan_aes_configuration_s          aes;
}t_ve_acoustic_ecan_configuration_u;
```

The elements of this structure are described below:

Element	Definition
ve_acoustic_ecan_id	An enumerated type used to define the type of acoustic ECAN to be used. The enumerated type, **t_ve_acoustic_ecan_id_e**, is defined in Appendix B.87 on Page 524.
ve_acoustic_ecan_config	Union of structures used to define the configuration structure to be used for this line ECAN. The available structures, selected by the *ve_acoustic_ecan_id* enumeration, is as follows: **t_ve_acoustic_ecan_aec_configuration_s** as described in Appendix B.11.1.1 on Page 267. **t_ve_acoustic_ecan_aec_complex_configuration_s** as described in Appendix B.11.1.2 on Page 268. **t_ve_acoustic_ecan_aes_configuration_s** as described in Appendix B.11.1.3 on Page 271.

There can only be one active VeriCall Edge Echo Canceller per-channel. If the Line Echo Canceller (defined in B.10.1.1 on Page 265) is enabled, it will take precedence over this Acoustic Echo Canceller, i.e. the Line Echo Canceller takes preference over the Acoustic Echo Canceller.

B.11.1.1 t_ve_acoustic_ecan_aec_configuration_s

The **t_ve_acoustic_ecan_aec_configuration_s** structure is used with the **t_ve_acoustic_ecan_configuration_s** structure as described in Appendix B.11 on Page 266. The structure allows for the simple configuration of the VeriCall Edge Acoustic Echo Canceller (AEC) algorithm task. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  t_ve_default_configuration_boolean_e   ecan_enable;
  UWord32                                configuration_profile;
  UWord32                                bulk_delay;
  t_ve_default_configuration_boolean_e   bulk_delay_estimator;
  Word32                                 search_range;
  Word16                                 post_gain;
  t_ve_default_configuration_boolean_e   write_audio;
  UWord32                                test_mode;
}t_ve_acoustic_ecan_aec_configuration_s;
```

The elements of this structure are described below:

Element	Definition
ecan_enable	Enumerated type to enable/disable the AEC. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521.
configuration_profile	Allows a set of pre-defined configurations to be applied to the AEC. Only values of 0, 1 and 2 should be used for this element. Each value should be tried in turn to find the best "fit".
bulk_delay	When the bulk delay estimator is disabled then this setting defines the bulk delay. When the bulk delay estimator is enabled then this setting defines the start of the search range (see below). The bulk delay must be greater than zero and the default value is 50 (milliseconds).
bulk_delay_estimator	Enumerated type to enable/disable the bulk delay estimator. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521.
search_range	The length of the search range for the bulk delay estimator. The start of the search range is defined by the bulk delay setting (see above). Default value is 50 (milliseconds).
post_gain	User defined integer value to allow post-gain scaling, before encoding, to be adjusted. This is a linear scaling value in Q8 format. The element defaults to no-scaling, 256. To double the volume this element should be set to 512.
write_audio	Enumerated type to enable/disable recording the AEC input and output to a series of files. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521. This should only be enabled to allow Trinity Convergence to analyze the data to provide tuning values. Defaults to disabled.
test_mode	Reserved for future use. This entry must be set to zero.

In order to optimally configure the AEC for specific environments, Trinity Convergence can supply specific values for this AEC configuration. In order to do this, the input and outputs of the AEC are recorded to a series of files (enabled by the `write_audio` structure element), which should then be sent to Trinity Convergence who will analyze the data and provide appropriate AEC configuration settings.

Continued >>

Typically, a number of files from different channels in applicable environments should be taken and supplied, this way the optimum values can be provided. The vendor should not attempt to interpret the information contained in these files.

For each AEC configured channel a series of files are written. The files to be supplied to Trinity Convergence are as follows:

- aec_fe_input_chan*x*.bin
- aec_ne_input_chan*x*.bin
- aec_output_chan*x*.bin
- aec_settings_chan*x*.txt

The *x* within the file names is replaced with the channel number the values were recorded from. These files can contain multiple calls and are written to the directory from which the VeriCall Edge system is executed. The files are overwritten if the VeriCall Edge system is restarted with the AEC enabled.

B.11.1.2 t_ve_acoustic_ecan_aec_complex_configuration_s

The **t_ve_acoustic_ecan_aec_complex_configuration_s** structure is used with the **t_ve_acoustic_ecan_configuration_s** structure as described in Appendix B.11 on Page 266. The structure allows for a more comprehensive configuration of the VeriCall Edge Acoustic Echo Canceller (AEC) algorithm task. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  t_ve_default_configuration_boolean_e    ecan_enable;
  Word32                                  nlp_level;
  Word32                                  nlp_hangover_1;
  Word32                                  nlp_hangover_2;
  Word32                                  nlp_hangover_3;
  Word32                                  nlp_hangover_4;
  Word32                                  adapt_mode;
  Word32                                  convergence;
  Word32                                  dt_thres1;
  Word32                                  dt_thres2;
  Word32                                  cng_level;
  Word32                                  delay_est;
  Word32                                  fselect;
  t_ve_default_configuration_boolean_e    hpflag;
  t_ve_default_configuration_boolean_e    update_farend_enable;
  UWord32                                 bulk_delay;
  t_ve_default_configuration_boolean_e    bulk_delay_estimator;
  Word32                                  search_range;
  Word16                                  post_gain;
  t_ve_default_configuration_boolean_e    write_audio;
  UWord32                                 test_mode;
}t_ve_acoustic_ecan_aec_complex_configuration_s;
```

Continued >>

The elements of the **t_ve_acoustic_ecan_aec_complex_configuration_s** structure are described below:

Element	Definition
ecan_enable	Enumerated type to enable/disable the AEC. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521.
nlp_level nlp_hangover_1 nlp_hangover_2 nlp_hangover_3 nlp_hangover_4 adapt_mode convergence dt_thres1 dt_thres2 cng_level delay_est fselect	These elements allow for AEC tuning and should initially be set to the default value of 0 (zero) by the vendor. Tuning values can be obtained from Trinity Convergence, the process for doing this is explained in the text below.
hpflag	Enumerated type to enable/disable the in-built AEC High Pass (HP) filter. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521. Defaults to disabled.
update_farend_enable	Enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521, which allows additional tuning of speaker output (from far-end) . This should only be used with care as adverse effects may occur.
bulk_delay	When the bulk delay estimator is disabled then this setting defines the bulk delay. When the bulk delay estimator is enabled then this setting defines the start of the search range (see below). The bulk delay must be greater than zero and the default value is 50 (milliseconds).
bulk_delay_estimator	Enumerated type to enable/disable the bulk delay estimator. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521.
search_range	The length of the search range for the bulk delay estimator. The start of the search range is defined by the bulk delay setting (see above). Default value is 50 (milliseconds).
post_gain	User defined integer value to allow post-gain scaling, before encoding, to be adjusted. This is a linear scaling value in Q8 format. The element defaults to no-scaling, 256.
write_audio	Enumerated type to enable/disable recording the AEC input and output to a series of files. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521. This should only be enabled to allow Trinity Convergence to analyze the data to provide tuning values. Defaults to disabled.
test_mode	Reserved for future use. This entry must be set to zero.

In order to optimally configure the AEC for specific environments, Trinity Convergence can supply specific values for this AEC configuration. In order to do this, the input and outputs of the AEC are recorded to a series of files (enabled by the `write_audio` structure element), which should then be sent to Trinity Convergence who will analyze the data and provide appropriate AEC configuration settings. Typically, a number of files from different channels in applicable environments should be taken and supplied, this way the optimum values can be provided. The vendor should not attempt to interpret the information contained in these files.

Continued >>

For each AEC configured channel a series of files are written. The files to be supplied to Trinity Convergence are as follows:

- aec_fe_input_chan*x*.bin
- aec_ne_input_chan*x*.bin
- aec_output_chan*x*.bin
- aec_settings_chan*x*.txt

The *x* within the file names is replaced with the channel number the values were recorded from. These files can contain multiple calls and are written to the directory from which the VeriCall Edge system is executed. The files are overwritten if the VeriCall Edge system is restarted with the AEC enabled.

B.11.1.3 t_ve_acoustic_ecan_aes_configuration_s

The **t_ve_acoustic_ecan_aes_configuration_s** structure is used with the **t_ve_acoustic_ecan_configuration_s** structure as described in Appendix B.11 on Page 266. The structure allows for the configuration of the VeriCall Edge Acoustic Echo Suppressor (AES) algorithm task. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  t_ve_default_configuration_boolean_e    ecan_enable;
  t_ve_default_configuration_boolean_e    prefilter_enable;
  t_ve_acoustic_ecan_aes_mode_e           mode;
  UWord32                                 bulk_delay;
  Word16                                  post_gain;
  t_ve_default_configuration_boolean_e    write_audio;
  UWord32                                 test_mode;
}t_ve_acoustic_ecan_aes_configuration_s;
```

The elements of this structure are described below:

Element	Definition
ecan_enable	Enumerated type to enable/disable the AEC. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521.
prefilter_enable	Enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521, which allows the pre-filter to be enabled or disabled.
mode	Enumerated type, **t_ve_acoustic_ecan_aes_mode_e** as described in Appendix B.88 on Page 525, which allows the echo suppression mode of operation to be defined.
bulk_delay	User defined integer value to set the bulk delay. The bulk delay must be greater than zero and the default value is 50 (milliseconds).
post_gain	User defined integer value to allow post-gain scaling, before encoding, to be adjusted. This is a linear scaling value in Q8 format. The element defaults to no-scaling, 256.
write_audio	Enumerated type to enable/disable recording the AEC input and output to a series of files. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521. This should only be enabled to allow Trinity Convergence to analyze the data to provide tuning values. Defaults to disabled.
test_mode	Reserved for future use. This entry must be set to zero.

In order to optimally configure the AEC for specific environments, Trinity Convergence can supply specific values for this AEC configuration. In order to do this, the input and outputs of the AES are recorded to a series of files (enabled by the `write_audio` structure element), which should then be sent to Trinity Convergence who will analyze the data and provide appropriate AES configuration settings. Typically, a number of files from different channels in applicable environments should be taken and supplied, this way the optimum values can be provided. The vendor should not attempt to interpret the information contained in these files.

Continued >>

For each AES configured channel a series of files are written. The files to be supplied to Trinity Convergence are as follows:

- aes_fe_input_chan*x*.bin
- aes_ne_input_chan*x*.bin
- aes_output_chan*x*.bin
- aes_settings_chan*x*.txt

The *x* within the file names is replaced with the channel number the values were recorded from. These files can contain multiple calls and are written to the directory from which the VeriCall Edge system is executed. The files are overwritten if the VeriCall Edge system is restarted with the AES enabled.

B.12 t_ve_gain_control_configuration_s

The **t_ve_gain_control_configuration_s** structure contains default configuration parameters for the Gain Control algorithm task. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  t_ve_default_configuration_boolean_e  enable;
  t_ve_gain_control_mode_e              mode;
  t_ve_gain_control_configuration_u     config;
}t_ve_gain_control_configuration_s;

typedef union
{
  t_ve_gain_control_agc_mode_e          agc_mode;
  Word32                                manual_gain;
}t_ve_gain_control_configuration_u;
```

The elements of this structure are described below:

Element	Definition
enable	Enumerated type to enable/disable gain control. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521.
mode	Enumerated type that determines the mode of gain control to be applied. The enumerated type, **t_ve_gain_control_mode_e**, is defined in Appendix B.91 on Page 528
config	A union of configuration elements that can be one of the following: agc_mode Enumerated type that determines the mode of automatic gain control. The enumerated type, **t_ve_gain_control_agc_mode_e** is defined in Appendix B.92 on Page 528 manual_gain The Manual Gain Control to be applied (in dB). Must be an integer in the range -24 to +24. Note: Applied gain is constant with Manual Gain Control and voice activated with Automatic Gain Control.

B.13 t_ve_ntd_algorithm_default_configuration_s

The **t_ve_ntd_algorithm_default_configuration_s** structure contains default configuration parameters for a Named Tone Detector algorithm task. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  t_ve_default_configuration_boolean_e          detect_2100_enable;
  t_ve_default_configuration_boolean_e          detect_1100_enable;
  t_ve_default_configuration_boolean_e          cp_enable;
  t_ve_ntd_country_type_default_configuration_e cp_country;
  t_ve_default_configuration_boolean_e          dtmf_enable;
  Word32                                        dtmf_normal_twist;
  Word32                                        dtmf_reverse_twist;
  Word32                                        dtmf_freq_spread;
}t_ve_ntd_algorithm_default_configuration_s;
```

The elements of this structure are detailed below:

Element	Definition
detect_2100_enable	Enumerated type to enable/disable the detection of 2100Hz tones. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521.
detect_1100_enable	Enumerated type to enable/disable the detection of 1100Hz tones. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521.
cp_enable	Enumerated type to enable/disable the detection of Call Progress tones. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521.
cp_country	Enumerated type to define the specific country Call Progress tones to be detected. The enumerated type, **t_ve_ntd_country_type_default_configuration_e**, is defined in Appendix B.93 on Page 529.
dtmf_enable	Enumerated type to enable/disable the detection of DTMF tones. The enumerated type, **t_ve_default_configuration_boolean_e**, is defined in Appendix B.84 on Page 521.
dtmf_normal_twist	Low frequency power, in dB, is greater than high frequency power. This is set to 8dB by default.
dtmf_reverse_twist	High frequency power, in dB, is greater than low frequency power. This is set to 4dB by default.
dtmf_freq_spread	Frequency spread percentage allowed in detecting DTMF tones. This is set to 1% by default.

B.14 t_ve_ntg_algorithm_default_configuration_s

The **t_ve_ntg_algorithm_default_configuration_s** structure contains default configuration parameters for a Named Tone Generator algorithm task. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  Word32     dtmf_level_dB;
  Word32     dtmf_duration_ms;
  Word32     dtmf_gap_duration_ms;
}t_ve_ntg_algorithm_default_configuration_s;
```

The elements of this structure are detailed below:

Element	Definition
dtmf_level_dB	The power level of DTMF tones to generate in dB. The default is -10dB with 0dB being full scale.
dtmf_duration_ms	The duration to play the DTMF tone in milliseconds. The default is 70ms.
dtmf_gap_duration_ms	The gap duration between DTMF tones in milliseconds. The default is 50ms.

Note that these levels are also used to determine the transmission characteristics of the digit strings used in calls to the VE_GenerateDigitString function.

B.15 t_ve_pre_proc_default_configuration_s

The **t_ve_pre_proc_default_configuration_s** structure determines the default configuration parameters for the VeriCall Edge pre-processor. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  t_ve_process_selector_default_configuration_e      type;
  t_ve_pre_proc_selector_default_configuration_u     config;
}t_ve_pre_proc_default_configuration_s;

typedef union{
  t_ve_null_pre_proc_default_configuration_s         null;
  t_ve_g711_pre_proc_default_configuration_s         g711;
  t_ve_video_pre_proc_default_configuration_s        video;
}t_ve_pre_proc_selector_default_configuration_u;
```

Continued >>

The elements of the **t_ve_pre_proc_default_configuration_s** structure are detailed below:

Element	Definition
type	Enumerated type, **t_ve_process_selector_default_configuration_e**, used to determine type of pre-processing to be used for this channel. The enumerated type is defined in Appendix B.83 on Page 521.
config	A union of one of the following structures: **t_ve_null_pre_proc_default_configuration_s** as described in Appendix B.15.1 on Page 276. **t_ve_g711_pre_proc_default_configuration_s** as described in Appendix B.15.2 on Page 277. **t_ve_video_pre_proc_default_configuration_s** as described in Appendix B.15.3 on Page 277.

B.15.1 t_ve_null_pre_proc_default_configuration_s

The **t_ve_null_pre_proc_default_configuration_s** structure provides a default configuration to be used within the VeriCall Edge system when there is no requirement for a pre-processor. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. This allows the user to select an end termination point for the stream without the need for a pre-processor. This may be required when, for example, the audio device is able to provide/accept 16-bit PCM Linear data. This structure is defined in **ve_configuration.h** and is described below.

```
typedef struct{
  UWord32      data_end_point_instance;
}t_ve_null_pre_proc_default_configuration_s;
```

The element of this structure is as follows:

Element	Definition
data_end_point_instance	Defines the end termination point for the stream. A data-end point instance number is associated to each VE_ATTRIBUTE_TYPE_DEFAULT_DATA_END_POINT attribute entry. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this instance.

B.15.2 t_ve_g711_pre_proc_default_configuration_s

The **t_ve_g711_pre_proc_default_configuration_s** structure contains default configuration parameters for a VeriCall Edge G.711 pre-processor algorithm task. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  t_ve_g711_default_mode_e     mode;
  Word32                       frame_size;
  Uword32                      data_end_point_instance;
}t_ve_g711_pre_proc_default_configuration_s;
```

The elements of this structure are detailed below:

Element	Definition
mode	The G.711 mode of operation enumerated type; **t_ve_g711_default_mode_e** is defined in Appendix B.85 on Page 522.
frame_size	The packet interval in milliseconds.
data_end_point_instance	The data-end point instance label that is associated to each VE_ATTRIBUTE_TYPE_DEFAULT_DATA_END_POINT attribute entry. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this instance.

B.15.3 t_ve_video_pre_proc_default_configuration_s

The **t_ve_video_pre_proc_default_configuration_s** structure contains configuration parameters for the VeriCall Edge video pre-processor algorithm task. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  /* Image from local camera to the display*/
  t_ve_video_default_local_image_enable_e   local_image_enable;
  UWord32                       local_image_h_origin;
  UWord32                       local_image_v_origin;
  UWord32                       local_image_h_size;
  UWord32                       local_image_v_size;
  UWord32                       local_image_h_scale_factor_num;
  UWord32                       local_image_h_scale_factor_den;
  UWord32                       local_image_v_scale_factor_num;
  UWord32                       local_image_v_scale_factor_den;
  t_ve_video_default_image_rotate_e         local_image_rotation;
  t_ve_video_default_image_mirror_e         local_image_mirror;
  t_ve_video_color_format_e     local_image_color_format;
  UWord32                       local_image_data_end_point_instance;
```

Continued >>

```
    /* Local camera settings */
    UWord32                          camera_h_size;
    UWord32                          camera_v_size;
    UWord32                          camera_data_end_point_instance;
    t_ve_video_color_format_e        camera_color_format;

    /* Image stream passed through to the encoder */
    UWord32                             tx_image_h_size;
    UWord32                             tx_image_v_size;
    UWord32                             tx_image_h_scale_factor_num;
    UWord32                             tx_image_h_scale_factor_den;
    UWord32                             tx_image_v_scale_factor_num;
    UWord32                             tx_image_v_scale_factor_den;
    t_ve_video_default_image_rotate_e   tx_image_rotation;
    t_ve_video_default_image_mirror_e   tx_image_mirror;
    t_ve_video_color_format_e           tx_image_color_format;
}t_ve_video_pre_proc_default_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
local_image_enable	Enumerated type, **t_ve_video_default_local_image_enable_e**, to activate or deactivate the displaying of the image from the local camera to the local display. The enumerated type is defined in Appendix B.94 on Page 530.
local_image_h_origin	Provides a relative offset to the top left hand corner of the display on the *x*-axis for the local image. The origin should be specified as an integer.
local_image_v_origin	Provides a relative offset to the top left hand corner of the display on the *y-axis* for the local image. The origin should be specified as an integer.
local_image_h_size	Specifies the width of the image that is sent from the local camera to the local display. The width should be specified as an integer.
local_image_v_size	Specifies the height of the image that is sent from the local camera to the local display. The height should be specified as an integer.
local_image_h_scale_factor_num	The horizontal scale factor numerator. Used in conjunction with local_image_h_scale_factor_den, it allows definition of the scale ratio for the horizontal width of the local image.
local_image_h_scale_factor_den	The horizontal scale factor denominator. Used in conjunction with local_image_h_scale_factor_num, it allows definition of the scale ratio for the horizontal width of the local image.
local_image_v_scale_factor_num	The vertical scale factor numerator. Used in conjunction with local_image_v_scale_factor_den, it allows definition of the scale ratio for the vertical height of the local image.
local_image_v_scale_factor_den	The vertical scale factor denominator. Used in conjunction with local_image_v_scale_factor_num, it allows definition of the scale ratio for the vertical height of the local image.
local_image_rotation	Enumerated type, **t_ve_video_default_image_rotate_e**, used to determine the rotation factor to be applied to the local image. The enumerated type is defined in Appendix B.95 on Page 530.

Continued >>

Element	Definition
local_image_mirror	Enumerated type, **t_ve_video_default_image_mirror_e**, used to determine the mirroring factor to be applied to the local image. The enumerated type is defined in Appendix B.96 on Page 531.
local_image_color_format	Enumerated type, **t_ve_video_color_format_e**, used to determine the color format of the local image. The enumerated type is defined in Appendix B.97 on Page 532.
local_image_data_end_point_instance	The local image data-end point instance. A data-end point instance number is associated to each VE_ATTRIBUTE_TYPE_DEFAULT_DATA_END_POINT attribute entry. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this instance.
camera_h_size	Determines the width of the image from the local camera that is written to the frame buffer (all received data is written to a frame buffer).
camera_v_size	Determines the height of the image from the local camera that is written to the frame buffer.
camera_data_end_point_instance	The camera data-end point instance. A data-end point instance number is associated to each VE_ATTRIBUTE_TYPE_DEFAULT_DATA_END_POINT attribute entry. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this instance.
camera_color_format	Enumerated type, **t_ve_video_color_format_e**, used to determine the color format of the local camera. The enumerated type is defined in Appendix B.97 on Page 532.
tx_image_h_size	The width of the image to be sent to the video encoder.
tx_image_v_size	The height of the image to be sent to the video encoder.
tx_image_h_scale_factor_num	The horizontal scale factor numerator. Used in conjunction with tx_image_h_scale_factor_den, it allows definition of the scale ratio for the horizontal width of the transmitted image.
tx_image_h_scale_factor_den	The horizontal scale factor denominator. Used in conjunction with `tx_image_h_scale_factor_num`, it allows definition of the scale ratio for the horizontal width of the transmitted image.
tx_image_v_scale_factor_num	The vertical scale factor numerator. Used in conjunction with tx_image_v_scale_factor_den, it allows definition of the scale ratio for the vertical height of the transmitted image.
tx_image_v_scale_factor_den	The vertical scale factor denominator. Used in conjunction with tx_image_v_scale_factor_num, it allows definition of the scale ratio for the vertical height of the transmitted image.
tx_image_rotation	Enumerated type, **t_ve_video_default_image_rotate_e**, used to determine the rotation factor to be applied to the image to be transmitted. The enumerated type is defined in Appendix B.95 on Page 530.
tx_image_mirror	Enumerated type, **t_ve_video_default_image_mirror_e**, used to determine the mirroring factor to be applied to the image to be transmitted. The enumerated type is defined in Appendix B.96 on Page 531.
tx_image_color_format	Enumerated type, **t_ve_video_color_format_e**, used to determine the color format of the image to be transmitted. The enumerated type is defined in Appendix B.97 on Page 532.

B.16 t_ve_post_proc_default_configuration_s

The **t_ve_post_proc_default_configuration_s** structure determines the default configuration parameters for the VeriCall Edge post-processor. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  t_ve_process_selector_default_configuration_e    type;
  t_ve_post_proc_selector_default_configuration_u   config;
}t_ve_post_proc_default_configuration_s;

typedef union{
  t_ve_null_post_proc_default_configuration_s      null;
  t_ve_g711_post_proc_default_configuration_s      g711;
  t_ve_video_post_proc_default_configuration_s     video;
}t_ve_post_proc_selector_default_configuration_u;
```

The elements of this structure are detailed below:

Element	Definition
type	Enumerated type, **t_ve_process_selector_default_configuration_e**, used to determine type of post-processing to be used for this channel. The enumerated type is defined in Appendix B.83 on Page 521.
config	A union of one of the following structures: **t_ve_null_post_proc_default_configuration_s** as described in Appendix B.16.1 on Page 281. **t_ve_g711_post_proc_default_configuration_s** as described in Appendix B.16.2 on Page 281. **t_ve_video_post_proc_default_configuration_s** as described in Appendix B.16.3 on Page 282.

B.16.1 t_ve_null_post_proc_default_configuration_s

The **t_ve_null_post_proc_default_configuration_s** structure provides a default configuration to be used within the VeriCall Edge system when there is no requirement for a post-processor. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. This allows the user to select an end termination point for the stream without the need for a post-processor. This structure is defined in **ve_configuration.h** and is described below.

```
typedef struct{
  UWord32          data_end_point_instance;
}t_ve_null_post_proc_default_configuration_s;
```

The element of the **t_ve_null_post_proc_default_configuration_s** structure is as follows:

Element	Definition
data_end_point_instance	Defines the end termination point for the stream. A data-end point instance number is associated to each VE_ATTRIBUTE_TYPE_DEFAULT_DATA_END_POINT attribute entry. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this instance.

In this version of the VeriCall Edge software, this configuration should only be applied for a PCM linear configuration.

B.16.2 t_ve_g711_post_proc_default_configuration_s

The **t_ve_g711_post_proc_default_configuration_s** structure contains default configuration parameters for a VeriCall Edge G.711 post-processor algorithm task. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  t_ve_g711_default_mode_e     mode;
  Uword32                      data_end_point_instance;
}t_ve_g711_post_proc_default_configuration_s;
```

The elements of this structure are detailed below:

Element	Definition
mode	The G.711 mode of operation enumerated type; **t_ve_g711_default_mode_e** is defined in Appendix B.85 on Page 522.
data_end_point_instance	The data-end point instance label that is associated to each VE_ATTRIBUTE_TYPE_DEFAULT_DATA_END_POINT attribute entry. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this instance.

B.16.3 t_ve_video_post_proc_default_configuration_s

The **t_ve_video_post_proc_default_configuration_s** structure contains default configuration parameters for the VeriCall Edge video post-processor algorithm task. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  UWord32                                rx_image_h_origin;
  UWord32                                rx_image_v_origin;
  UWord32                                rx_image_h_size;
  UWord32                                rx_image_v_size;
  UWord32                                rx_image_h_scale_factor_num;
  UWord32                                rx_image_h_scale_factor_den;
  UWord32                                rx_image_v_scale_factor_num;
  UWord32                                rx_image_v_scale_factor_den;
  t_ve_video_color_format_e              rx_image_color_format;
  t_ve_video_default_filter_selection_e  rx_image_filter_selection;
  t_ve_video_default_image_rotate_e      rx_image_rotation;
  t_ve_video_default_image_mirror_e      rx_image_mirror;
  UWord32                          rx_image_data_end_point_instance;
}t_ve_video_post_proc_default_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
rx_image_h_origin	Provides a relative offset to the top left hand corner of the display on the *x*-axis for the received image. The origin should be specified as an integer.
rx_image_v_origin	Provides a relative offset to the top left hand corner of the display on the *y-axis* for the received image. The origin should be specified as an integer.
rx_image_h_size	Specifies the width of the received image to be displayed. The width should be specified as an integer.
rx_image_v_size	Specifies the height of the received image to be displayed. The height should be specified as an integer.
rx_image_h_scale_factor_num	The horizontal scale factor numerator. Used in conjunction with rx_image_h_scale_factor_den, it allows definition of the scale ratio for the horizontal width of the received image to be displayed.
rx_image_h_scale_factor_den	The horizontal scale factor denominator. Used in conjunction with rx_image_h_scale_factor_num, it allows definition of the scale ratio for the horizontal of the received image to be displayed.
rx_image_v_scale_factor_num	The vertical scale factor numerator. Used in conjunction with rx_image_v_scale_factor_den, it allows definition of the scale ratio for the vertical height of the received image to be displayed.
rx_image_v_scale_factor_den	The vertical scale factor denominator. Used in conjunction with rx_image_v_scale_factor_num, it allows definition of the scale ratio for the vertical height of the received image to be displayed.

Continued >>

Element	Definition
rx_image_color_format	Enumerated type, **t_ve_video_color_format_e**, used to determine the color format of the received image. The enumerated type is defined in Appendix B.97 on Page 532.
rx_image_filter_selection	Enumerated type, **t_ve_video_filter_selection_e**, used to allow selection of de-ringing and de-blocking functionality of the received image. The enumerated type is defined in Appendix B.98 on Page 532.
rx_image_rotation	Enumerated type, **t_ve_video_default_image_rotate_e**, used to determine the rotation factor to be applied to the received image. The enumerated type is defined in Appendix B.95 on Page 530.
rx_image_mirror	Enumerated type, **t_ve_video_default_image_mirror_e**, used to determine the mirroring factor to be applied to the received image. The enumerated type is defined in Appendix B.96 on Page 531.
rx_image_data_end_point_instance	The received image data-end point instance. A data-end point instance number is associated to each VE_ATTRIBUTE_TYPE_DEFAULT_DATA_END_POINT attribute entry. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this instance.

B.17 t_ve_sip_default_configuration_s

The **t_ve_sip_default_configuration_s** structure is used for the VeriCall Edge system configuration to define the SIP configuration parameters. Table C.1 on Page 659 shows the VeriCall Edge attribute associated with this structure. This structure is defined in **ve_sip_configuration.h** and is described below:

```
typedef struct{
  t_ve_sip_state_flag_default_configuration_e  sip_stack_state;
  UWord16       num_transactions_per_channel;
  UWord16       num_transaction_users_per_channel;
  UWord16       num_presence_users_per_user;
  UWord16       num_mw_users_per_user;
  UWord16       num_dialog_users_per_user;
  UWord16       num_dialogs_per_channel;
  UWord16       num_sip_message_storage_per_transaction;
  UWord16       num_extra_sip_message_storage;
  UWord16       num_event_timers_per_user;
  UWord16       sip_port;
  Word16        sip_tcp_port;
  Word16        sip_tcp_tls_port;
  t_ve_sip_symmetric_mode_default_configuration_e
                                                  sip_symmetric_mode;
  UWord32       cseq_number;
  UWord32       max_forwards;
  UWord32       expires_time;
  UWord32       max_call_bandwidth_kbits_per_sec;
  t_ve_sip_stack_liberalisation_s      liberalisation;
  t_ve_sip_stack_feature_support_s     feature_support;
  t_ve_sip_routing_s                   routing;
  Word8           user_agent[VE_SIP_MAX_USER_AGENT_STRING_SIZE];
}t_ve_sip_default_configuration_s;
```

Continued >>

The elements of the **t_ve_sip_default_configuration_s** structure are as follows:

Element	Definition
sip_stack_state	An enumerated type, **t_ve_sip_state_flag_default_configuration_e** as described in Appendix B.183 on Page 605, that acts as a flag to enable/disable the VeriCall Edge SIP stack.
num_transactions_per_channel	The number of transactions available per channel. Because transactions can persist after their original use has ended, before being freed for reuse, spare transactions should be made available. Transactions are used as a pool across all available channels, e.g. 2 channels with 10 transactions configured per channel will result in 20 transactions being available for use by both channels. Default is 10 per channel.
num_transactions_users_per_channel	The number of transaction users available per channel. Transaction users are used, for example, throughout the lifetime of a call or registration. Transaction users are used as a pool across all channels, e.g. 2 channels with 10 transaction users per channel will allocate a pool of 20 transaction users for use by both channels. Default is 10 per channel.
num_presence_users_per_user	The number of presence users (buddies) available per user. Presence users are used as a pool across all channels, e.g. 2 channels with 10 presence users per channel will allocate a pool of 20 presence users for use by both channels. Default is 10 per user.
num_mw_users_per_user	The number of message waiting subscription users (mailboxes) available per user. Message Waiting (mw) users are used as a pool across all channels, e.g. 2 users with 10 mw users per user will allocate a pool of 20 mw users for use by both users. Default is 2 per user.
num_dialog_users_per_user	The number of dialog subscription users available per user. Dialog users are used as a pool across all channels, e.g. 2 users with 10 dialog users per user will allocate a pool of 20 dialog users for use by both users.
num_dialogs_per_channel	The number of available dialogs per channel. Dialogs are used when a call is initiated and freed when the call is terminated. By default, this entry is set to be 2.
num_sip_message_storage_per_transaction	The amount of SIP messages the SIP stack can internally store, the SIP messages are associated with the transactions. By default, this entry is set to be 2.
num_extra_sip_message_storage	The number of extra SIP messages the SIP stack can store internally in order to cope with responding to inbound SIP messages when it is heavily loaded. By default, this entry is set to be 4.
num_event_timers_per_user	The number of event timers to associate with each user. Event timers are used, for example, to terminate an outgoing INVITE that has been cancelled if no response if received in a designated time. By default this entry is set to be 0.
sip_port	The port the SIP stack should listen to for inbound UDP SIP traffic. By default this is port 5060.
sip_tcp_port	The port the SIP stack should listen to for inbound TCP SIP traffic. By default this is port 5060. Setting this value to be zero will stop a TCP port from being created.
sip_tcp_tls_port	The port the SIP stack should listen to for inbound TCP TLS SIP traffic. By default this is port 5061.
sip_symmetric_mode	The enumerated SIP mode of operation to be used, which can be either asymmetric or symmetric. The enumerated type, **t_ve_sip_symmetric_mode_default_configuration_e**, is described in Appendix B.182 on Page 604.

	• Symmetric mode means that SIP messages originate from port 5060. • Asymmetric mode means SIP messages originate from a random port chosen by the OS.
cseq_number	The start value of the SIP Command Sequence (Cseq) header integer field. This can be set to any reasonable value. By default, this entry is set to be 100.
max_forwards	Used to determine the maximum number of times a SIP packet can be forwarded before being discarded. By default, this entry is set to be 70.
expires_time	The expire time requested when sending s REGISTER message to a SIP registrar. Used to determine how long the URI is valid for. This is measured in seconds and by default, it is set to be 3600.
max_call_bandwidth_kbits_per_sec	The maximum per-call bandwidth that the VeriCall Edge system will negotiate. This field is measured in kbps where k is defined as 1000 bits. If this value is set to zero, bandwidth negotiation will be disabled. By default this value is set to be zero.
liberalisation	A structure, **t_ve_sip_stack_liberalisation_s** described in Appendix B.40 on Page 443, containing flags to make the VeriCall Edge SIP stack less strict in certain circumstances. This element is reserved for future use and will currently be ignored by the SIP stack.
feature_support	A structure, **t_ve_sip_stack_feature_support_s** described in Appendix B.41 on Page 443, containing flags to enable/disable certain features in the VeriCall Edge SIP stack. It may be necessary to disable certain features when encountering issues with problematic third party hardware.
routing	A structure, **t_ve_sip_routing_s** described in Appendix B.42 on Page 445, containing routing settings for the VeriCall Edge SIP stack.
user_agent	A string representation of the SIP User Agent header to be used. This string must not exceed `VE_SIP_MAX_USER_AGENT_STRING_SIZE` in length including the termination character. If the user application does not set this element, the header in SIP messages appears as: **User-Agent: VeriCall Edge** It the user application sets the string to, for example, the text "hello" then the header in SIP messages appears as: **User-Agent: hello (VeriCall Edge)**

B.18 t_ve_tx_lip_sync_default_configuration_s

The **t_ve_tx_lip_sync_default_configuration_s** structure is used for the VeriCall Edge system configuration to define system audio to video lip-synchronization default configuration for the VeriCall Edge system. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. This structure is defined in **ve_configuration.h** and is described below:

```
typedef struct{
  Word32     tx_video_to_audio_msec_configuration;
}t_ve_tx_lip_sync_default_configuration_s;
```

The element of this structure is as follows:

Element	Definition
tx_video_to_audio_msec_configuration	Defines the system-wide lip-synchronization default configuration for system defined associated audio and video channels. The value is defined in milliseconds.

B.19 t_ve_data_end_point_default_configuration_s

The **t_ve_data_end_point_default_configuration_s** structure is used to define audio and video data-end points within the VeriCall Edge system, Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. This structure is defined in **ve_configuration.h** and is described below:

```
typedef struct{
  t_ve_data_end_point_default_type_e      type;
  Word8      descriptor[VE_END_POINT_DESCRIPTOR_STRING];
}t_ve_data_end_point_default_configuration_s;
```

The structure elements are as follows:

Element	Definition
type	Indicates the data-end point default type and is an enumerated type of **t_ve_data_end_point_default_type_e** as described in Appendix B.99 on Page 533.
descriptor	A label identifier for this data-end point. The label identifier should be unique in value and should be used within channel configurations that use this defined data-end point. The label identifier must not exceed `VE_END_POINT_DESCRIPTOR_STRING` in length (currently defined as being 100 characters including termination character).

Video data-end point instances will generally be either a camera or display and hence will never be represented by a duplex data-end point.

B.20 t_ve_video_tear_protect_default_configuration_s

The **t_ve_video_tear_protect_default_configuration_s** structure is used to enable or disable video tear protection within a video defined channel. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. This structure is defined in **ve_configuration.h** and is described below:

```
typedef struct{
  t_ve_default_configuration_boolean_e    enable;
}t_ve_video_tear_protect_default_configuration_s;
```

The structure elements are as follows:

Element	Definition
type	Enumerated type, **t_ve_default_configuration_boolean_e** as described in Appendix B.84 on Page 521, allowing the VeriCall Edge video tear protection to be enabled or disabled.

B.21 t_ve_channel_s

The **t_ve_channel_s** structure is used for the VeriCall Edge system configuration channel definition, allowing per-channel settings to be assembled. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. This structure is defined in **ve_channel.h** and is described below:

```
typedef struct{
  t_ve_channel_type_config_e           channel_type;
  t_ve_channel_config_boolean_e        line_ecan_enable;
  t_ve_channel_config_boolean_e        acoustic_ecan_enable;
  t_ve_channel_config_boolean_e        gain_control_enable;
  t_ve_channel_config_boolean_e        dtmf_relay_enable;
  t_ve_channel_config_boolean_e        named_tone_detector_enable;
  t_ve_channel_config_boolean_e        named_tone_generator_enable;
  t_ve_channel_3_way_calling_config_s  three_way_calling;
  t_ve_channel_data_configuration_s    data_config;
  t_ve_channel_packet_configuration_s  packet_config;
}t_ve_channel_s;
```

The elements of this structure are as follows:

Element	Definition
channel_type	Enumerated type **t_ve_channel_type_config_e** as defined in Appendix B.102 on Page 535, to determine whether the channel under consideration is to be configured for audio or video.
line_ecan_enable	Enumerated type **t_ve_channel_config_boolean_e** as defined in Appendix B.101 on Page 534, to determine whether the channel under consideration is to have an ECAN enabled.
acoustic_ecan_enable	Enumerated type **t_ve_channel_config_boolean_e** as defined in Appendix B.101 on Page 534, to determine whether the channel under consideration is to have an Acoustic ECAN enabled.
gain_control_enable;	Enumerated type **t_ve_channel_config_boolean_e** as defined in Appendix B.101 on Page 534, to determine whether the channel under consideration is to have Gain Control enabled.
dtmf_relay_enable	Enumerated type **t_ve_channel_config_boolean_e** as defined in Appendix B.101 on Page 534, to determine whether the channel under consideration is to have DTMF relay enabled.
named_tone_detector_enable	Enumerated type **t_ve_channel_config_boolean_e** as defined in Appendix B.101 on Page 534, to determine whether the channel under consideration is to have NTD enabled.
named_tone_generator_enable	Enumerated type **t_ve_channel_config_boolean_e** as defined in Appendix B.101 on Page 534, to determine whether the channel under consideration is to have NTG enabled.
three_way_calling	Structure of type **t_ve_channel_3_way_calling_config_s**, as defined in Appendix B.21.1 on Page 290, to allow this channel to have the ability to perform 3-way calling.
data_config	Structure of type **t_ve_channel_data_configuration_s**, as defined in Appendix B.21.2 on Page 291, to determine the channel data-side configuration.
packet_config	Structure of type **t_ve_channel_packet_configuration_s**, as defined in Appendix B.21.3 on Page 293, to determine the channel packet-side configuration.

B.21.1 t_ve_channel_3_way_calling_config_s

The **t_ve_channel_3_way_calling_config_s** structure is part of the **t_ve_channel_s** structure defined in Appendix B.21 on Page 289. The structure is used to determine whether 3-way calling is to be enabled on this channel or not. This structure is defined in **ve_channel.h** and is described below:

```
typedef struct{
  t_ve_channel_config_boolean_e     three_way_calling_is_active;
}t_ve_channel_3_way_calling_config_s;
```

The elements of this structure are as follows:

Element	Definition
three_way_calling_is_active	An enumerated value of type **t_ve_channel_config_boolean_e** as defined in Appendix B.101 on Page 534, specifying whether 3-way calling is to be enabled or disabled within this channel.

If 3-way calling is enabled within a channel, a primary and secondary element of the channel is created. The primary element is referred to as the primary channel while the secondary element is referred to as the secondary-channel, sometimes also called the virtual channel. Both these sub-channels that make a channel need to be configured.

B.21.2 t_ve_channel_data_configuration_s

The **t_ve_channel_data_configuration_s** structure is part of the **t_ve_channel_s** structure defined in Appendix B.21 on Page 289. The structure is used to configure the channel data-side parameters. This structure is defined in **ve_channel.h** and is described below:

```
typedef struct{
  t_ve_channel_data_format_e         channel_data_format;
  t_ve_channel_pre_conditioning_s    channel_pre_conditioning;
  t_ve_channel_post_conditioning_s   channel_post_conditioning;
  UWord32                            num_source_data_end_points;
  UWord32 channel_source_data_end_point[VE_NUM_CHANNEL_END_POINTS];
  UWord32                            num_sink_data_end_points;
  UWord32 channel_sink_data_end_point[VE_NUM_CHANNEL_END_POINTS];
}t_ve_channel_data_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
channel_data_format	An enumerated value of type **t_ve_channel_data_format_e**, as defined in Appendix B.100 on Page 534, specifying the format of the data-side data.
channel_pre_conditioning	A structure of type **t_ve_channel_pre_conditioning_s** as defined in Appendix B.21.2.1 on Page 292. Used to determine pre-processing options for this channel.
channel_post_conditioning	A structure of type **t_ve_channel_post_conditioning_s** as defined in Appendix B.21.2.2 on Page 292. Used to determine post-processing options for this channel.
num_source_data_end_points	The number of data-end points for the source of this channel.
channel_source_data_end_point	The data-end point for the source of this channel. The maximum number of source data-end points definable for this channel cannot exceed `VE_NUM_CHANNEL_END_POINTS` (currently defined as being 10).
num_sink_data_end_points	The number of data-end points for the sink of this channel.
channel_sink_data_end_point	The data-end point for the sink of this channel. The maximum number of sink data-end points definable for this channel cannot exceed `VE_NUM_CHANNEL_END_POINTS` (currently defined as being 10).

B.21.2.1 t_ve_channel_pre_conditioning_s

The **t_ve_channel_pre_conditioning_s** structure is used within the **t_ve_channel_data_configuration_s** structure as defined in Appendix B.21.2 on Page 291. The **t_ve_channel_pre_conditioning_s** structure defines pre-processing options for the specified channel. This structure is defined in **ve_channel.h** and is described below:

```
typedef struct{
  Word32     reserved;
}t_ve_channel_pre_conditioning_s;
```

The element of this structure is as follows:

Element	Definition
reserved	Reserved for future use.

B.21.2.2 t_ve_channel_post_conditioning_s

The **t_ve_channel_post_conditioning_s** structure is used within the **t_ve_channel_data_configuration_s** structure as defined in Appendix B.21.2 on Page 291. The **t_ve_channel_post_conditioning_s** structure defines post-processing options for the specified channel. This structure is defined in **ve_channel.h** and is described below:

```
typedef struct{
  t_ve_high_pass_filter_state_e     hp_filter_state;
}t_ve_channel_post_conditioning_s;
```

The element of this structure is as follows:

Element	Definition
hp_filter_state	An enumerated type, **t_ve_high_pass_filter_state_e** as defined on in Appendix B.103 on Page 535, which enables or disables the post-processing high-pass filter.

This structure is reserved for future use and should not be used within this release of VeriCall Edge.

B.21.3 t_ve_channel_packet_configuration_s

The **t_ve_channel_packet_configuration_s** structure is part of the **t_ve_channel_s** structure defined in Appendix B.21 on Page 289. The structure is used to define the packet side configuration parameters of a VeriCall Edge channel. This structure is defined in **ve_channel.h** and is described below:

```
typedef struct{
  t_ve_channel_port_configuration_s    primary;
  t_ve_channel_port_configuration_s    secondary;
  UWord32                              ip_address;
  UWord32                              media_capability_list_entries;
  t_ve_media_id_e
            media_capability_list[VE_MAX_MEDIA_CAPABILITY_LIST];
}t_ve_channel_packet_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
primary	A structure of type **t_ve_channel_port_configuration_s** as defined in Appendix B.21.3.1 on Page 294. This entry allows the port configuration parameters of this primary channel to be defined.
secondary	A structure of type **t_ve_channel_port_configuration_s** as defined in Appendix B.21.3.1 on Page 294. This entry allows the port configuration parameters of this secondary channel to be defined.
ip_address	This element is reserved for future use and must be set to zero.
media_capability_list_entries	The number of media capabilities defined in this packet side channel configuration.
media_capability_list	Array of media type capability algorithms that this packet-side channel can accommodate. The entries are of the enumerated type **t_ve_media_id_e** as defined in Appendix B.110 on Page 539. The maximum number of media capabilities the channel is capable of handling cannot exceed `VE_MAX_MEDIA_CAPABILITY_LIST`, Which is defined from the MAX_MEDIA_ID element of the enumerated type **t_ve_media_type_e** as described in Appendix B.106 on Page 537.

The VeriCall Edge system uses the network byte ordering for IP addresses when a UWord32 value is required.

B.21.3.1 t_ve_channel_port_configuration_s

The **t_ve_channel_port_configuration_s** structure is part of the **t_ve_channel_packet_configuration_s** structure defined in Appendix B.21.3 on Page 293. The structure is used to define the packet side port configuration parameters of a VeriCall Edge primary or secondary channel. The **t_ve_channel_port_configuration_s** structure is defined in **ve_channel.h** and described below:

```
typedef struct{
  UWord16                          rtp_port;
  UWord16                          rtcp_port;
  t_ve_channel_config_boolean_e    jitter_buffer_enable;
  t_ve_channel_config_boolean_e    rtcp_port_state;
  t_ve_channel_config_boolean_e    port_address_check_state
  t_ve_channel_config_boolean_e    ssrc_validation_state;
}t_ve_channel_port_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
rtp_port	Defines the RTP port number to be used with this primary or secondary channel.
rtcp_port	Defines the RTCP port number to be used with this primary or secondary channel.
jitter_buffer_enable	An enumerated type, **t_ve_channel_type_config_e** as defined in Appendix B.102 on Page 535. The enumerated type allows for the VeriCall Edge adaptive jitter buffer to be enabled or disabled on this primary or secondary channel.
rtcp_port_state	An enumerated type, **t_ve_channel_type_config_e** as defined in Appendix B.102 on Page 535. The enumerated type allows the RTCP port to be activated or deactivated.
port_address_check_state	An enumerated type, **t_ve_channel_type_config_e** as defined in Appendix B.102 on Page 535. The enumerated type enables validation of the IP address for RTP/RTCP.
ssrc_validation_state	An enumerated type, **t_ve_channel_type_config_e** as defined in Appendix B.102 on Page 535. The enumerated type enables or disables checking of the SSRC.

B.22 t_ve_callback_channel_data_configuration_u

The **t_ve_callback_channel_data_configuration_u** union is used in calls to the function that was registered using the **VE_RegisterChannelDataDeviceConfigure** function. The union allows for configuration of audio and video capabilities of the user defined VeriCall Edge Data Manager callbacks. This structure is defined in **ve_callback.h** and is described below:

```
typedef union{
  t_ve_callback_audio_data_channel_config_s    audio_config;
  t_ve_callback_video_data_channel_config_s    video_config;
}t_ve_callback_data_channel_config_u;
```

The structures of this union are as follows:

Element	Definition
audio_config	A structure, **t_ve_callback_audio_data_channel_config_s**, as defined in Appendix B.22.1 on Page 296, specifies the audio configuration to be used with this callback function.
video_config	A structure, **t_ve_callback_video_data_channel_config_s**, as defined in Appendix B.22.2 on Page 296, specifies the video configuration to be used with this callback function.

B.22.1 t_ve_callback_audio_data_channel_config_s

The **t_ve_callback_audio_data_channel_config_s** structure is part of the **t_ve_callback_channel_data_configuration_u** union as defined in Appendix B.22 on Page 295. The structure is used to configure the audio channel to be used with this VeriCall Edge callback function. This structure is defined in **ve_callback.h** and is described below:

```
typedef struct{
  UWord32                              sample_rate;
  UWord32                              buffer_depth;
  t_ve_callback_audio_endian_e         endian;
}t_ve_callback_audio_data_channel_config_s;
```

The elements of this structure are as follows:

Element	Definition
sample_rate	The sample rate of the data to be transmitted or received, in Hz.
buffer_depth	The buffer depth of the audio channel to be configured.
endian	An enumerated value of type **t_ve_callback_audio_endian_e**, as defined in Appendix B.173 on Page 596, specifying the Endian type to be used for data transfer.

B.22.2 t_ve_callback_video_data_channel_config_s

The **t_ve_callback_video_data_channel_config_s** structure is part of the **t_ve_callback_channel_data_configuration_u** union as defined in Appendix B.22 on Page 295. The structure is used to configure the audio channel to be used with this VeriCall Edge callback function. This structure is defined in **ve_callback.h** and is described below:

```
typedef struct{
  t_ve_callback_video_color_format_e   format;
  UWord32                              frame_rate;
  UWord32                              h_size;
  UWord32                              v_size;
}t_ve_callback_video_data_channel_config_s;
```

The elements of this structure are as follows:

Element	Definition
format	An enumerated type, **t_ve_callback_video_color_format_e** as define in Appendix B.175 on Page 598. The enumerated type allows for the configuration of the channels video color format mode of operation.
frame_rate	The frame rate of the video image. This value of this entry will be dependent on the camera used but is normally a maximum of 30fps.
h_size	The horizontal size of the video image. The maximum size of this entry is dependent on the display used.
v_size	The vertical size of the video image. The maximum size of this entry is dependent on the display used.

B.23 t_ve_callback_network_open_s

The **t_ve_callback_network_open_s** structure is used in calls to the function that was registered using the **VE_RegisterPacketTransportOpen** function. The **t_ve_callback_network_open_s** structure allows for the configuration of the packet network to be used with the user defined VeriCall Edge Data Manager callback. This structure is defined in **ve_callback.h** and is described below:

```
typedef struct{
  t_ve_callback_protocol_e     protocol;
  t_ve_callback_direction_e    direction;
}t_ve_callback_network_open_s;
```

The elements of this structure are as follows:

Element	Definition
protocol	An enumerated type, **t_ve_callback_protocol_e** as defined in Appendix B.171 on Page 595. The enumerated type allows for the configuration of the packet network protocol that is to be used.
direction	An enumerated type, **t_ve_callback_direction_e** as defined in Appendix B.172 on Page 596. The enumerated type allows for the definition of whether the registered user defined VeriCall Edge callback function is for the reception or transmission of data.

B.24 t_ve_stream_configuration_s

The **t_ve_stream_configuration_s** structure is used in calls to the **VE_StartStream**, **VE_UpdateStream** and **VE_SIPSessionStart_WithConfig** API functions. The structure is also part of the **t_ve_user_msg_sip_session_started_s** structure as described in Appendix B.33.1 on Page 360. The structure is written by the VeriCall Edge system to inform the user application of the media configuration capabilities of the remote end. This structure is defined in **ve_defs.h** and is described below.

```
typedef struct{
  UWord32            num_media_descriptors;
  t_ve_media_description_s
                     media_descriptor[VE_MAX_MEDIA_DESCRIPTORS];
  UWord32            call_bandwidth_kbits_per_sec;
}t_ve_stream_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
num_media_descriptors	The number of media descriptors within the VeriCall Edge system.
media_descriptor	Details of the remote connection configuration structure, **t_ve_media_description_s**, as defined in Appendix B.27 on Page 302. This allows multiple media descriptors to be defined, e.g. audio and video configurations for an associated audio/video call. The number of media descriptors cannot exceed `VE_MAX_MEDIA_DESCRIPTORS` which is currently defined as being 10
call_bandwidth_kbits_per_sec	The maximum bandwidth available for this stream to use. If there is insufficient bandwidth available for the media type being used, the steam may be dropped. This is measured in 1000 * bps.

Note that the media descriptors in the **t_ve_stream_configuration_s** structure relate to the user channel(s) on a 1:1 basis. This means that if the user is associated with channel *n* and *n*+1, then media descriptor 1 refers to channel *n* and media descriptor 2 refers to channel *n*+1.

B.25 t_ve_user_handle_s

The **t_ve_user_handle_s** structure is used by a large number of the VeriCall Edge API functions as well as some of the VeriCall API associated structures. The **t_ve_user_handle_s** structure contains the user's identification handle and is used to identify the VeriCall Edge user in API calls. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The **t_ve_user_handle_s** structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  UWord32    handle;
  UWord32    route_application_id
}t_ve_user_handle_s;
```

The elements of this structure are as follows:

Element	Definition
handle	The handle that identifies the specified user.
route_application_id	The application ID of the application to which messages are to be routed. For internal use only.

B.26 t_ve_user_channel_details_s

The **t_ve_user_channel_details_s** structure is used by calls to the **VE_AddUser** and **VE_UpdateUser** functions. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The structure is used to configure the channel details of a particular user to be registered or updated with the VeriCall Edge system. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  UWord32                    number_of_channels;
  t_ve_user_channel_association_s
                  channel_association[VE_MAX_CHANNEL_ASSOCIATION];
}t_ve_user_channel_details_s;
```

The elements of this structure are as follows:

Element	Definition
number_of_channels	The number of channels associated with this user.
channel_association	Structure of type **t_ve_user_channel_association_s** to define the per channel algorithm preference associated with this user. The structure is defined in Appendix B.26.1 on Page 301. The number of channels that can be associated with this user cannot exceed VE_MAX_CHANNEL_ASSOCIATION that is currently defined as being 2.

Please note that this structure refers to a VeriCall Edge channel. Primary and secondary channels are components of a VeriCall Edge channel.

B.26.1 t_ve_user_channel_association_s

The **t_ve_user_channel_association_s** structure is part of the **t_ve_user_channel_details_s** structure as defined in Appendix B.26 on Page 300. The structure is used to define a channels number and algorithm preferences. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_user_channel_selection_s     channel_selection;
  t_ve_media_id_e    alg_preference[VE_MAX_NUM_ALG_PREF_LIST];
  UWord32            num_algs_in_pref_list;
}t_ve_user_channel_association_s;
```

The elements of this structure are as follows:

Element	Definition
channel_selection	Structure of type **t_ve_user_channel_selection_s** as described in Appendix B.26.1.1 on Page 301. This structure allows the definition of the channel number and optional sub-channel type to be associated with a user.
alg_preference	Array of sequential algorithm priority order preference types that this channel is capable of negotiating. The first entry within this array is the most preferred choice. The entries within this array should be the enumerated algorithm Media ID, as defined in **t_ve_media_id_e** described in Appendix B.110 on Page 539. The maximum number of algorithm types is defined by VE_MAX_NUM_ALG_PREF_LIST which is currently defined as being the MAX_MEDIA_ID element of the enumerated type **t_ve_media_type_e** as described in Appendix B.106 on Page 537 .
num_algs_in_pref_list	The number of algorithms defined in the `alg_preference` array.

B.26.1.1 t_ve_user_channel_selection_s

The **t_ve_user_channel_selection_s** structure is part of the **t_ve_user_channel_association_s** structure as defined in Appendix B.26.1 on Page 301. The structure is used to define the channel number, and optionally define the sub-channel type, associated with a user. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  UWord32                          channel_number;
  t_ve_user_sub_channel_select_e   sub_channel_select;
}t_ve_user_channel_selection_s;
```

The elements of this structure are as follows:

Element	Definition
channel_number	The channel number to be associated with a user.
sub_channel_select	An enumerated type, **t_ve_user_sub_channel_select_e** as described in Appendix B.167 on Page 593, used to determine the sub-channel type to be associated with a user.

B.27 t_ve_media_description_s

The **t_ve_media_description_s** structure is part of the **t_ve_stream_configuration_s** structure as described in Appendix B.24 on Page 298. The **t_ve_media_description_s** structure provides configuration information on a number of remote connection media capabilities. The **t_ve_media_description_s** structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  UWord32                              channel;
  t_ve_user_sub_channel_select_e       sub_channel_select;
  t_ve_media_field_s                   media_field;
  t_ve_media_stream_direction_e        direction;
  t_ve_media_ip_configuration_s        ip_configuration;
  t_ve_media_attrib_mapping_s          rtpmapping_attrib;
}t_ve_media_description_s;
```

The elements of this structure are as follows:

Element	Definition
channel	The channel identifier (number).
sub_channel_select	An enumerated type, **t_ve_user_sub_channel_index_e** as described in Appendix B.168 on Page 594, that determines the sub-channel type use with this media description.
media_field	Structure of type **t_ve_media_field_s** as described in Appendix B.27.1 on Page 303.
direction	An enumerated type, **t_ve_media_stream_direction_e** as described in Appendix B.130 on Page 557, that indicates the direction (transmit, receive or both)
ip_configuration	Structure of type **t_ve_media_ip_configuration_s** as described in Appendix B.27.2 on Page 304.
rtpmapping_attrib	Structure of type **t_ve_media_attrib_mapping_s** as described in Appendix B.27.3 on Page 305.

B.27.1 t_ve_media_field_s

The **t_ve_media_field_s** structure is part of the **t_ve_media_description_s** structure as described in Appendix B.27 on Page 302. The **t_ve_media_field_s** structure provides information on the remote connection capabilities. The **t_ve_media_field_s** structure is defined in **ve_media.h** and is described below:

```
typedef struct{
  t_ve_media_type_e              media_type;
  UWord16                        remote_base_port;
  UWord16                        remote_base_rtcp_port;
  UWord16                        number_of_remote_ports;
  t_ve_media_transport_type_e    transport_type;
}t_ve_media_field_s;
```

The elements of this structure are as follows:

Element	Definition
media_type	The enumerated media type, **t_ve_media_type_e** as described in Appendix B.106 on Page 537.
remote_base_port	The base port number of the remote connection.
remote_base_rtcp_port	The base RTCP port number of the remote connection.
number_of_remote_ports	The number of remote ports used for this connection. Reserved for future use.
transport_type	The enumerated transport type, **t_ve_media_transport_type_e** as described in Appendix B.105 on Page 536.

B.27.2 t_ve_media_ip_configuration_s

The **t_ve_media_ip_configuration_s** structure is part of the **t_ve_media_description_s** structure as described in Appendix B.27 on Page 302. Table C.1 on Page 659 shows the VeriCall Edge attribute associated with this structure. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_media_ip_configuration_e     ip_configuration;
  t_ve_media_ip_specification_u     ip_specification;
}t_ve_media_ip_configuration_s;

typedef union{
  Word8      ip_domain[VE_MAX_MEDIA_IP_ADDRESS_SIZE];
  UWord32    ip_address;
}t_ve_media_ip_specification_u;
```

The elements of the **t_ve_media_ip_configuration_s** structure are as follows:

Element	Definition
ip_configuration	The version of IP addressing used by the VeriCall Edge system. The IP version is an enumerated value of type **t_ve_media_ip_configuration_e**. The IP version types are described in Appendix B.107 on Page. 537.
ip_specification	A union of one of the following: **`ip_domain`** A string representation of the IP address which cannot exceed `VE_MAX_MEDIA_IP_ADDRESS_SIZE` characters in length (currently defined as being 100 characters including the termination character). **`ip_address`** A hexadecimal representation of the IP address.

The VeriCall Edge system uses the network byte ordering for IP addresses when a UWord32 value is required.

B.27.3 t_ve_media_attrib_mapping_s

The **t_ve_media_attrib_mapping_s** structure is part of the **t_ve_media_description_s** structure as described in Appendix B.27 on Page 302. The **t_ve_media_attrib_mapping_s** structure provides configuration information on a number of remote connection media capabilities. The **t_ve_media_attrib_mapping_s** structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  UWord32                          num_rtp_maps;
  UWord32                          default_rtp_map;
  t_ve_media_attrib_mapping_u      rtpmap;
}t_ve_media_attrib_mapping_s;

typedef union{
  t_ve_media_rtp_attributes_s
                                rtpmap[VE_MAX_NUMBER_OF_RTP_MAPPINGS];
}t_ve_media_attrib_mapping_u;
```

The elements of this structure are as follows:

Element	Definition
num_rtp_maps	The number of available **t_ve_media_rtp_attributes_s** structures used in the VeriCall Edge system.
default_rtp_map	The default RTP mapping to be used.
rtpmap	Union of structures (which cannot exceed `VE_MAX_NUMBER_OF_RTP_MAPPINGS` which is currently defined as being 10) which can be one of the following: **t_ve_media_rtp_attributes_s** as described in Appendix B.27.3.1 on Page 306.

B.27.3.1 t_ve_media_rtp_attributes_s

The **t_ve_media_rtp_attributes_s** structure is part of the **t_ve_media_attrib_mapping_s** structure as described in Appendix B.27.3 on Page 305. The **t_ve_media_rtp_attributes_s** structure provides configuration information on the selected remote connection media capabilities. The **t_ve_media_rtpmaps_s** structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_media_id_e                    media_id;
  UWord32                            clock_rate;
  UWord32                            payload_type;
  t_ve_media_rtp_configuration_u     rtp_config;
}t_ve_media_rtp_attributes_s;
```

The elements of this structure are as follows:

Element	Definition
media_id	The ID of the media profile type, as defined in the **t_ve_media_id_e** enumeration described in Appendix B.110 on Page 539.
clock_rate	The packet interval in milliseconds. Reserved for future use.
payload_type	The payload type number. There are a number of "well known" payload types that VeriCall Edge will try and match with. If no match can be found VeriCall Edge will try with the given payload type.
rtp_config	A union of structures defining the RTP configuration. The union **t_ve_media_rtp_configuration_u** is described in Appendix B.28 on Page 307.

B.28 t_ve_media_rtp_configuration_u

The **t_ve_media_rtp_configuration_u** union of structures is part of structure **t_ve_media_rtp_attributes_s** as defined in Appendix B.27.3.1 on Page 306, and structure **t_ve_h323_call_update_attributes_s** as defined in Appendix B.56.3 on Page 473. The **t_ve_media_rtp_configuration_u** union defines an RTP configuration. The union is defined in **ve_defs.h** and described below:

```
typedef union{
  t_ve_media_g711_configuration_s      g711;
  t_ve_media_g729a_configuration_s     g729a;
  t_ve_media_g723_1_configuration_s    g723;
  t_ve_media_g726_configuration_s      g726;
  t_ve_media_gsm_amr_configuration_s   gsm_amr;
  t_ve_media_ilbc_configuration_s      ilbc;
  t_ve_media_h263_configuration_s      h263;
  t_ve_media_mpeg4_configuration_s     mpeg4;
}t_ve_media_rtp_configuration_u;
```

The elements of this union of structures are as follows:

Element	Definition
g711	A structure **t_ve_media_g711_configuration_s** as defined in Appendix B.28.1 on Page 308 that defines the G.711 algorithm configuration.
g729a	A structure **t_ve_media_g729a_configuration_s** as defined in Appendix B.28.2 on Page 308 that defines the G.729A algorithm configuration.
g723	A structure **t_ve_media_g723_1_configuration_s** as defined in Appendix B.28.3 on Page 309 that defines the G.723 algorithm configuration.
g726	A **structure t_ve_media_g726_configuration_s** as defined in Appendix B.28.4on Page 309 that defines the G.726 algorithm configuration.
gsm_amr	A structure **t_ve_media_gsm_amr_configuration_s** as defined in Appendix B.28.5 on Page 310 that defines the GSM-AMR algorithm configuration.
ilbc	A structure **t_ve_media_ilbc_configuration_s** as defined in Appendix B.28.6 on Page 312 that defines the iLBC algorithm configuration.
h263	A structure **t_ve_media_h263_configuration_s** as defined in Appendix B.28.7 on Page 313 that defines the H.263 algorithm configuration.
mpeg4	A structure **t_ve_media_mpeg4_configuration_s** as defined in Appendix B.28.8 on Page 314 that defines the MPEG-4 algorithm configuration.

B.28.1 t_ve_media_g711_configuration_s

The **t_ve_media_g711_configuration_s** structure is part of the **t_ve_media_rtp_configuration_u** union of structures as defined in Appendix B.28 on Page 307. The **t_ve_media_g711_configuration_s** structure defines media configuration options for a G.711 algorithm task. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  UWord32    frame_size;
}t_ve_media_g711_configuration_s;
```

The element of this structure is as follows:

Element	Definition
frame_size	The packet interval in milliseconds (with VAD enabled the frame size must be a multiple of 10ms).

B.28.2 t_ve_media_g729a_configuration_s

The **t_ve_media_g729a_configuration_s** structure is part of the **t_ve_media_rtp_configuration_u** union of structures as defined in Appendix B.28 on Page 307. The **t_ve_media_g729a_configuration_s** structure defines media configuration options for a G.729A algorithm task. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_media_vad_state_e     vad_state;
  UWord32                    frame_size;
}t_ve_media_g729a_configuration_s;
```

The element of this structure is as follows:

Element	Definition
vad_state	An enumerated type, **t_ve_media_vad_state_e** as defined on in Appendix B.121 on Page 549, which enables or disables the VAD capability of the G.729.
frame_size	The packet interval in milliseconds (with VAD enabled the frame size must be a multiple of 10ms).

B.28.3 t_ve_media_g723_1_configuration_s

The **t_ve_media_g723_1_configuration_s** structure is part of the **t_ve_media_rtp_configuration_u** union of structures as defined in Appendix B.28 on Page 307. The **t_ve_media_g723_1_configuration_s** structure defines media configuration options for a G.723 algorithm task. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_media_vad_state_e     vad_state;
  t_ve_media_g723_1_rate_e   rate;
  UWord32                    frame_size;
}t_ve_media_g723_1_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
vad_state	An enumerated type, **t_ve_media_vad_state_e** as defined on in Appendix B.121 on Page 549, which enables or disables the VAD capability of the G.723.
rate	An enumerated type, **t_ve_media_g723_1_rate_e** as defined on in Appendix B.122 on Page 550, which configures the G.723 data rate.
frame_size	The packet interval in milliseconds (with VAD enabled the frame size must be a multiple of 10ms.

B.28.4 t_ve_media_g726_configuration_s

The **t_ve_media_g726_configuration_s** structure is part of the **t_ve_media_rtp_configuration_u** union of structures as defined in Appendix B.28 on Page 307. The **t_ve_media_g726_configuration_s** structure defines media configuration options for a G.726 algorithm task. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  UWord32      frame_size;
}t_ve_media_g726_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
frame_size	The packet interval in milliseconds.

B.28.5 t_ve_media_gsm_amr_configuration_s

The **t_ve_media_gsm_amr_configuration_s** structure is part of the **t_ve_media_rtp_configuration_u** union of structures as defined in Appendix B.28 on Page 307. The structure defines media configuration options for a GSM-AMR algorithm task. This structure is defined in **ve_defs.h** and described below:

```
typedef struct{
  t_ve_media_gsm_amr_vad_state_e     vad_state;
  t_ve_media_gsm_amr_rate_e          gsm_amr_rate;
  UWord32                            frame_size;
  UWord16                            octet_align;
  UWord16                            mode_set;
  UWord16                            mode_change_period;
  UWord16                            mode_change_neighbor;
  UWord16                            crc;
  UWord16                            robust_sorting;
  UWord16                            interleaving;
  UWord16                            channels;
}t_ve_media_gsm_amr_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
vad_state	An enumerated type, **t_ve_media_gsm_amr_vad_state_e** as defined in Appendix B.129 on Page 556, which enables or disables the VAD capability of the GSM-AMR algorithm.
gsm_amr_rate	An enumerated type, **t_ve_media_gsm_amr_rate_e** as defined in Appendix B.126 on Page 553, which configures the GSM-AMR data rate.
frame_size	The packet interval in milliseconds (must be a multiple of 20ms).
octet_align	Determines RTP packing scheme. Can be set to 1 (octet-aligned operation) or 0 (bandwidth efficient operation).
mode_set	A bit field that restricts the active set of codec modes. The bit-field word can be composed of the following: 0x01 4.75 kbps 0x02 5.15 kbps 0x04 5.9 kbps 0x08 6.7 kbps 0x10 7.4 kbps 0x20 7.95kbps 0x40 10.2 kbps 0x80 12.2 kbps
mode_change_period	The interval at which codec mode changes are allowed (in frames).
mode_change_neighbor	Determines allowable mode changes . Can be set to 1 (changes to neighboring modes in active codec mode set only allowed) or 0 (all mode changes allowed).
crc	Determines whether CRCs are included in payload. Can be set to 1 (included) or 0 (not included). Setting of 1 also implies octet-aligned operation.
robust_sorting	Determines payload sorting. Can be set to 1 (robust sorting) or 0 (simple sorting).

Element	Definition
interleaving	Defines the maximum number of frame-blocks allowed in an interleaving group and implies octet-aligned operation. If set to zero then interleaving not used.
channels	The number of audio channels (must be set to 1).

B.28.6 t_ve_media_ilbc_configuration_s

The **t B.28.6t_ve_media_ilbc_configuration_s** structure is part of the **t_ve_media_rtp_configuration_u** union of structures as defined in Appendix B.28 on Page 307. The structure defines media configuration options for an iLBC algorithm task. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  UWord32     frame_size;
}t_ve_media_ilbc_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
frame_size	The packet interval in milliseconds.

B.28.7 t_ve_media_h263_configuration_s

The **t_ve_media_h263_configuration_s** structure is part of the **t_ve_media_rtp_configuration_u** union of structures as defined in Appendix B.28 on Page 307. The **t_ve_media_h263_configuration_s** structure defines media configuration options for an H.263 video algorithm task. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_media_video_profile_selection_e   enc_profile;
  t_ve_media_video_profile_level_e       enc_level;
  t_ve_media_video_encoding_mode_rate_s  i_frame_rate;
  UWord32                                enc_bit_rate;
  UWord32                                frame_rate;
  UWord32                                IntraPerVop_percentage;
}t_ve_media_h263_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
enc_profile	An enumerated type, **t_ve_media_video_profile_selection_e** as defined on in Appendix B.114 on Page 543, which determines the encoder profile to be used.
enc_level	An enumerated type, **t_ve_media_video_profile_level_e** as defined on in Appendix B.119 on Page 548, which determines the encoder profile level to be used.
i_frame_rate	A structure, **t_ve_media_video_encoding_mode_rate_s** as defined in Appendix B.29.23 on Page 341, which determines the rate at which I-Frames are transmitted.
enc_bit_rate	Defines the encoding transmit rate. The VeriCall Edge encoder can support any integer rate up to 400000bps, however, only the following rates are currently validated: 32000bps 64000bps 128000bps 384000bps Value to be entered the full rate value, e.g. 384000 for 384kbps (where k represents 1000).
frame_rate	The H.263 frame rate in frames-per-second (fps).
IntraPerVop_percentage	Allows the Cyclic Intra Refresh rate (I-blocks) to be set as a percentage of the number of macro blocks in the current picture size. The range is 0 to 100 where 0 means use the internal default setting. For example, setting the value to be 10 means 10% of the macro blocks of the current picture size are I-blocks.

B.28.8 t_ve_media_mpeg4_configuration_s

The **t_ve_media_mpeg4_configuration_s** structure is part of the **t_ve_media_rtp_configuration_u** union of structures as defined in Appendix B.28 on Page 307. The **t_ve_media_mpeg4_configuration_s** structure defines media configuration options for an MPEG-4 video algorithm task. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_media_video_encoding_mode_rate_s        i_frame_rate;
  t_ve_media_video_quantization_param_e        quant_param;
  t_ve_media_video_cyclic_intra_refresh_s      cir;
  t_ve_media_video_pid_rate_control_s          pid_rate_cntrl;
  t_ve_media_video_simple_profile_e            profile_and_level;
  t_ve_media_video_mpeg4_stream_container_s    stream_container;
}t_ve_media_mpeg4_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
i_frame_rate	A structure, **t_ve_media_video_encoding_mode_rate_s** as defined in Appendix B.29.23 on Page 341, which determines the rate at which I-Frames are transmitted.
quant_param	An enumerated type, **t_ve_media_video_quantization_param_e** as described in Appendix B.116 on Page 544, that determines the Quantization Parameter (QP) to be used during the encoding sequence.
cir	A structure, **t_ve_media_video_cyclic_intra_refresh_s** as described in Appendix B.29.16.4 on Page 331, that allows control of the Cyclic Intra Refresh (CIR) values.
pid_rate_cntrl	A structure, **t_ve_media_video_pid_rate_control_s** as described in Appendix B.29.16.5 on Page 332, which allows the configuration of the MPEG-4 encoder rate control parameters.
profile_and_level	An enumerated type, **t_ve_media_video_simple_profile_e** as described in Appendix B.118 on Page 547, that determines the simple MPEG-4 profile to be used.
stream_container	A structure, **t_ve_media_video_mpeg4_stream_container_s** as described in Appendix B.29.24 on Page 342, which defines the configuration ASCII string for the MPEG-4 complete header.

B.29 t_ve_media_definition_s

The **t_ve_media_definition_s** structure is used for VeriCall Edge system configuration to define the media types that can be handled by the VeriCall Edge Media Engine. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. This structure is defined in **ve_media.h** and described below:

```
typedef struct{
  t_ve_media_id_e              media_id;
  t_ve_media_data_type_e       media_data_type;
  t_ve_media_configuration_u   ve_config_enc;
  t_ve_media_configuration_u   ve_config_dec;
  UWord16                      payload_type;
  UWord32                      clock_rate;
}t_ve_media_definition_s;

typedef union{
  t_ve_media_g711_vad_encoder_algorithm_configuration_s g711_enc;
  t_ve_media_g711_vad_decoder_algorithm_configuration_s g711_dec;
  t_ve_media_g729a_encoder_algorithm_configuration_s    g729_enc;
  t_ve_media_g729a_decoder_algorithm_configuration_s    g729_dec;
  t_ve_media_g723_1_encoder_algorithm_configuration_s   g723_1_enc;
  t_ve_media_g723_1_decoder_algorithm_configuration_s   g723_1_dec;
  t_ve_media_g726_encoder_algorithm_configuration_s     g726_enc;
  t_ve_media_g726_decoder_algorithm_configuration_s     g726_dec;
  t_ve_media_gsm_amr_encoder_algorithm_configuration_s
                                                        gsm_amr_enc;
  t_ve_media_gsm_amr_decoder_algorithm_configuration_s
                                                        gsm_amr_dec;
  t_ve_media_cn_encoder_algorithm_configuration_s        cn_enc;
  t_ve_media_cn_decoder_algorithm_configuration_s        cn_dec;
  t_ve_media_te_algorithm_configuration_s               te_enc;
  t_ve_media_te_algorithm_configuration_s               te_dec;
  t_ve_media_video_encoder_h263_algorithm_configuration_s
                                                        video_h263_enc;
  t_ve_media_video_decoder_h263_algorithm_configuration_s
                                                        video_h263_dec;
  t_ve_media_video_encoder_mpeg4_algorithm_configuration_s
                                                       video_mpeg4_enc;
  t_ve_media_video_decoder_mpeg4_algorithm_configuration_s
                                                       video_mpeg4_dec;
  t_ve_media_eg711_encoder_algorithm_configuration_s    eg711_enc;
  t_ve_media_eg711_decoder_algorithm_configuration_s    eg711_dec;
  t_ve_media_ilbc_encoder_algorithm_configuration_s     ilbc_enc;
  t_ve_media_ilbc_decoder_algorithm_configuration_s     ilbc_dec;
}t_ve_media_configuration_u;
```

Continued >>

The elements of the **t_ve_media_definition_s** structure are as follows:

Element	Definition
media_id	Enumerated media identifier type. The enumeration type, **t_ve_media_id_e** is defined in Appendix B.110 on Page 539.
media_data_type	Enumerated type of media data. The enumeration type, **t_ve_media_data_type_e** is defined in Appendix B.104 on Page 536.
ve_config_enc	Union of encode configurations for this media stream. These structures are listed in the table below.
ve_config_dec	Union of decode configurations for this media stream. These structures are listed in the table below.
payload_type	RTP payload type for decode (if applicable). The payload type values are as defined in the appropriate algorithm RFC/ITU Standard.
clock_rate	The transmit clock rate to be used for this media definition.

Union of Structures	Reference
t_ve_media_g711_vad_encoder_algorithm_configuration_s	Appendix B.29.1 on Page 317.
t_ve_media_g711_vad_decoder_algorithm_configuration_s	Appendix B.29.2 on Page 317.
t_ve_media_g729a_encoder_algorithm_configuration_s	Appendix B.29.3 on Page 318.
t_ve_media_g729a_decoder_algorithm_configuration_s	Appendix B.29.4 on Page 318.
t_ve_media_g723_1_encoder_algorithm_configuration_s	Appendix B.29.5 on Page 319.
t_ve_media_g723_1_decoder_algorithm_configuration_s	Appendix B.29.6 on Page 319.
t_ve_media_g726_encoder_algorithm_configuration_s	Appendix B.29.7 on Page 320.
t_ve_media_g726_decoder_algorithm_configuration_s	Appendix B.29.8 on Page 320D.
t_ve_media_gsm_amr_encoder_algorithm_configuration_s	Appendix B.29.7 on Page 320.
t_ve_media_gsm_amr_decoder_algorithm_configuration_s	Appendix B.29.10 on Page 323.
t_ve_media_cn_encoder_algorithm_configuration_s	Appendix B.29.11 on Page 324.
t_ve_media_cn_decoder_algorithm_configuration_s	Appendix B.29.12 on Page 324.
t_ve_media_te_algorithm_configuration_s	Appendix B.29.13 on Page 325.
t_ve_media_video_encoder_h263_algorithm_configuration_s	Appendix B.29.14 on Page 325.
t_ve_media_video_decoder_h263_algorithm_configuration_s	Appendix B.29.15 on Page 326.
t_ve_media_video_encoder_mpeg4_algorithm_configuration_s	Appendix B.29.16 on Page 327.
t_ve_media_video_decoder_mpeg4_algorithm_configuration_s	Appendix B.29.17 on Page 336.
t_ve_media_eg711_encoder_algorithm_configuration_s	Appendix B.29.19 on Page 337.
t_ve_media_eg711_decoder_algorithm_configuration_s	Appendix B.29.20 on Page 338.
t_ve_media_ilbc_encoder_algorithm_configuration_s	Appendix B.29.21 on Page 339.
t_ve_media_ilbc_decoder_algorithm_configuration_s	Appendix B.29.22 on Page 340.

B.29.1 t_ve_media_g711_vad_encoder_algorithm_configuration_s

The **t_ve_media_g711_vad_encoder_algorithm_configuration_s** structure contains configuration parameters for a G.711 encoder Voice Activity Detector (VAD) algorithm task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  t_ve_media_g711_mode_e            encoder_mode;
  t_ve_media_vad_enable_boolean_e   vad_enabled;
  Word32                            frame_size;
}t_ve_media_g711_vad_encoder_algorithm_configuration_s;
```

The elements of the G.711 Encoder Configuration with VAD structure are described below:

Element	Definition
encoder_mode	The G.711 mode of operation enumerated type; **t_ve_media_g711_mode_e** is defined in Appendix B.108 on Page 538.
vad_enabled	Enumerated type, **t_ve_media_vad_enable_boolean_e**, to enable/disable the VAD. The enumerated type is defined in B.109 on Page 538.
frame_size	The packet interval in milliseconds (with VAD enabled the frame size must be a multiple of 10ms up to, and including, a maximum of 40ms).

B.29.2 t_ve_media_g711_vad_decoder_algorithm_configuration_s

The **t_ve_media_g711_vad_decoder_algorithm_configuration_s** structure contains configuration parameters for a G.711 decoder Voice Activity Detector (VAD) algorithm task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  t_ve_media_g711_mode_e     decoder_mode;
}t_ve_media_g711_vad_decoder_algorithm_configuration_s;
```

The element of this structure is described below:

Element	Definition
decoder_mode	The G.711 mode of operation enumerated type; **t_ve_media_g711_mode_e** is defined in Appendix B.108 on Page 538.

A decoder configuration is required because G.711 a-Law and G.711 µ-Law cannot be differentiated from the packets received.

B.29.3 t_ve_media_g729a_encoder_algorithm_configuration_s

The **t_ve_media_g729a_encoder_algorithm_configuration_s** structure contains configuration parameters for a G.729A encoder algorithm task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  t_ve_media_vad_enable_boolean_e   vad_enable;
  Word32                            frame_size;
}t_ve_media_g729a_encoder_algorithm_configuration_s;
```

The elements of this structure are described below:

Element	Definition
vad_enable	Enumerated type, **t_ve_media_vad_enable_boolean_e**, to enable/disable the VAD. The enumerated type is defined in B.109 on Page 538.
frame_size	The packet interval in milliseconds (with VAD enabled the frame size must be a multiple of 10ms up to, and including, a maximum of 40ms).

B.29.4 t_ve_media_g729a_decoder_algorithm_configuration_s

The **t_ve_media_g729a_decoder_algorithm_configuration_s** structure contains configuration parameters for a G.729A decoder algorithm task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  Word32     reserved;
}t_ve_media_g729a_decoder_algorithm_configuration_s;
```

The element of this structure is as follows:

Element	Definition
reserved	Reserved for future use.

B.29.5 t_ve_media_g723_1_encoder_algorithm_configuration_s

The **t_ve_media_g723_1_encoder_algorithm_configuration_s** structure contains configuration parameters for a G.723.1 encoder algorithm task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  t_ve_media_g723_1_mode_e                  encoderMode;
  t_ve_media_g723_1_hpf_enable_boolean_e    hp_filter_enable;
  t_ve_media_vad_enable_boolean_e           vad_enable;
  Word32                                    frame_size;
}t_ve_media_g723_1_encoder_algorithm_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
encoderMode	An enumerated type, **t_ve_media_g723_1_mode_e** as defined on in Appendix B.123 on Page 551, which configures the G.723 encoder data rate.
hp_filter_enable	An enumerated type, **t_ve_media_g723_1_hpf_enable_boolean_e** as defined on in Appendix B.124 on Page 551, which allows the enablement of the G.723 encoder high-pass filter.
vad_enable	Enumerated type, **t_ve_media_vad_enable_boolean_e**, to enable/disable the VAD. The enumerated type is defined in B.109 on Page 538.
frame_size	The packet interval in milliseconds.

B.29.6 t_ve_media_g723_1_decoder_algorithm_configuration_s

The **t_ve_media_g723_1_decoder_algorithm_configuration_s** structure contains configuration parameters for a G.723.1 decoder algorithm task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  t_ve_media_g723_1_pf_enable_boolean_e     pf_enable;
  Word32                                    frame_size;
}t_ve_media_g723_1_decoder_algorithm_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
pf_enable	An enumerated type, **t_ve_media_g723_1_pf_enable_boolean_e** as defined on in Appendix B.125 on Page 552, which allows the enablement of the G.723 decoder post filter.
frame_size	The packet interval in milliseconds.

B.29.7 t_ve_media_g726_encoder_algorithm_configuration_s

The **t_ve_media_g726_encoder_algorithm_configuration_s** structure contains configuration parameters for a G.726 encoder algorithm task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  Word32              frame_size;
}t_ve_media_g726_encoder_algorithm_configuration_s;
```

The element of this structure is as follows:

Element	Definition
frame_size	The packet interval in milliseconds.

B.29.8 t_ve_media_g726_decoder_algorithm_configuration_s

The **t_ve_media_g726_decoder_algorithm_configuration_s** structure contains configuration parameters for a G.726 decoder algorithm task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  Word32              frame_size;
}t_ve_media_g726_decoder_algorithm_configuration_s;
```

The element of this structure is as follows:

Element	Definition
frame_size	The packet interval in milliseconds.

B.29.9 t_ve_media_gsm_amr_encoder_algorithm_configuration_s

The **t_ve_media_gsm_amr_encoder_algorithm_configuration_s** structure contains configuration parameters for a GSM-AMR encoder algorithm task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  t_ve_media_gsm_amr_mode_e          encoderMode;
  t_ve_media_gsm_amr_vad_mode_e      dtx;
  UWord32                            frame_size;
  UWord16                            octet_align;
  UWord16                            mode_set;
  UWord16                            mode_change_period;
  UWord16                            mode_change_neighbor;
  UWord16                            crc;
  UWord16                            robust_sorting;
  UWord16                            interleaving;
  UWord16                            channels;

}t_ve_media_gsm_amr_encoder_algorithm_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
encoderMode	An enumerated type, **t_ve_media_gsm_amr_mode_e** as defined on in Appendix B.127 on Page 554, which configures the GSM-AMR encoder data rate.
dtx	An enumerated type, **t_ve_media_gsm_amr_vad_mode_e** as defined on in Appendix B.128 on Page 555, which configures the GSM-AMR encoder discontinuous transmission mode of operation.
frame_size	The packet interval in milliseconds (must be a multiple of 20ms).
octet_align	Determines RTP packing scheme. Can be set to 1 (octet-aligned operation) or 0 (bandwidth efficient operation).
mode_set	A bit field that restricts the active set of codec modes. The bit-field word can be composed of the following: 0x01 4.75 kbps 0x02 5.15 kbps 0x04 5.9 kbps 0x08 6.7 kbps 0x10 7.4 kbps 0x20 7.95kbps 0x40 10.2 kbps 0x80 12.2 kbps
mode_change_period	The interval at which codec mode changes are allowed (in frames).
mode_change_neighbor	Determines allowable mode changes . Can be set to 1 (changes to neighboring modes in active codec mode set only allowed) or 0 (all mode changes allowed).

Continued >>

Element	Definition
crc	Determines whether CRCs are included in payload. Can be set to 1 (included) or 0 (not included). Setting of 1 also implies octet-aligned operation.
robust_sorting	Determines payload sorting. Can be set to 1 (robust sorting) or 0 (simple sorting).
interleaving	Defines the maximum number of frame-blocks allowed in an interleaving group and implies octet-aligned operation. If set to zero then interleaving not used.
channels	The number of audio channels (must be set to 1).

B.29.10 t_ve_media_gsm_amr_decoder_algorithm_configuration_s

The **t_ve_media_gsm_amr_decoder_algorithm_configuration_s** structure contains configuration parameters for a GSM-AMR decoder algorithm task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct
  {
     UWord16     octet_align;
     UWord16     crc;
     UWord16     robust_sorting;
     UWord16     interleaving;
  }t_ve_media_gsm_amr_decoder_algorithm_configuration_s;
```

The element of this structure is as follows:

Element	Definition
octet_align	Determines RTP packing scheme. Can be set to 1 (octet-aligned operation) or 0 (bandwidth efficient operation).
crc	Determines whether CRCs are included in payload. Can be set to 1 (included) or 0 (not included). Setting of 1 also implies octet-aligned operation.
robust_sorting	Determines payload sorting. Can be set to 1 (robust sorting) or 0 (simple sorting).
interleaving	Defines the maximum number of frame-blocks allowed in an interleaving group and implies octet-aligned operation. If set to zero then interleaving not used.

B.29.11 t_ve_media_cn_encoder_algorithm_configuration_s

The **t_ve_media_cn_encoder_algorithm_configuration_s** structure contains configuration parameters for a Comfort Noise Generator (CNG) algorithm encoder task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  UWord32    reserved;
}t_ve_media_cn_encoder_algorithm_configuration_s;
```

The element of this structure is as follows:

Element	Definition
reserved	Reserved for future use.

B.29.12 t_ve_media_cn_decoder_algorithm_configuration_s

The **t_ve_media_cn_decoder_algorithm_configuration_s** structure contains configuration parameters for a Comfort Noise Generator (CNG) algorithm decoder task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_configuration.h** and described below:

```
typedef struct{
  t_ve_media_g711_mode_e     g711_decoder_mode;
}t_ve_media_cn_decoder_algorithm_configuration_s;
```

The single element of this structure is described below:

Element	Definition
g711_decoder_mode	The G.711 mode of operation enumerated type; **t_ve_media_g711_mode_e** is defined in Appendix B.108 on Page 538.

B.29.13 t_ve_media_te_algorithm_configuration_s

The **t_ve_media_te_algorithm_configuration_s** structure contains configuration parameters for a Telephony Event (TE) algorithm task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. This structure is used to configure the algorithm for an encoder and decoder. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  UWord32    reserved;
}t_ve_media_te_algorithm_configuration_s;
```

The element of this structure is as follows:

Element	Definition
reserved	Reserved for future use.

B.29.14 t_ve_media_video_encoder_h263_algorithm_configuration_s

The **t_ve_media_video_encoder_h263_algorithm_configuration_s** structure contains configuration parameters for the VeriCall Edge H.263 video encoder and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  t_ve_media_video_profile_selection_e    enc_profile;
  t_ve_media_video_profile_level_e        enc_level;
  t_ve_media_video_encoding_mode_e        enc_mode;
  t_ve_media_video_frame_type_selection_s enc_frame_type_selection;
  UWord32                                 enc_bit_rate;
  UWord32                                 frame_rate;
  UWord32                                 IntraPerVop_percentage;
}t_ve_media_video_encoder_h263_algorithm_configuration_s;
```

Continued >>

The elements of the **t_ve_media_video_encoder_h263_algorithm_configuration_s** structure are as follows:

Element	Definition
enc_profile	An enumerated type, **t_ve_media_video_profile_selection_e** as defined on in Appendix B.114 on Page 543, which determines the encoder profile to be used.
enc_level	An enumerated type, **t_ve_media_video_profile_level_e** as defined on in Appendix B.119 on Page 548, which determines the encoder profile level to be used.
enc_mode	Enumerated type, **t_ve_media_video_encoding_mode_e**, which allows the selection of the supported encoding modes. The enumerated type is defined in Appendix B.113 on Page 542.
enc_frame_type_selection	Structure of type **t_ve_media_video_frame_type_selection_s** as described in Appendix B.29.16 on Page 327. Use of this element defines whether the encoder should support only I-Frames or both I-Frames and P-Frames.
enc_bit_rate	Defines the encoding transmit rate. The VeriCall Edge encoder can support any integer rate up to 400000bps, however, only the following rates are currently validated: 32000bps 64000bps 128000bps 384000bps Value to be entered the full rate value, e.g. 384000 for 384kbps (where k represents 1000).
frame_rate	The input frame rate to the encoder. This will normally be dictated by the frame rate of the camera.
IntraPerVop_percentage	Allows the Cyclic Intra Refresh rate (I-blocks) to be set as a percentage of the number of macro blocks in the current picture size. The range is 0 to 100 where 0 means use the internal default setting. For example, setting the value to be 10 means 10% of the macro blocks of the current picture size are I-blocks. The default value is: `VE_MEDIA_DEFAULT_H263_ENC_INTRAPERVOP_PERCENTAGE` = 10

B.29.15 t_ve_media_video_decoder_h263_algorithm_configuration_s

The **t_ve_media_video_decoder_h263_algorithm_configuration_s** structure contains configuration parameters for the VeriCall Edge H.263 video decoder and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  UWord32    reserved;
}t_ve_media_video_decoder_h263_algorithm_configuration_s;
```

The element of this structure is as follows:

Element	Definition
reserved	Reserved for future use.

B.29.16 t_ve_media_video_encoder_mpeg4_algorithm_configuration_s

The **t_ve_media_video_encoder_mpeg4_algorithm_configuration_s** structure contains configuration parameters for the VeriCall Edge MPEG-4 video encoder and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. and is performed at the VeriCall Edge commit time. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  t_ve_media_video_quantization_param_e     quant_param;
  t_ve_media_video_config_state_e           hec_state;
  t_ve_media_video_user_data_s              video_seq_header;
  t_ve_media_video_user_data_s              video_object;
  t_ve_media_video_user_data_s              video_object_layer;
  t_ve_media_video_user_data_s              group_of_vop;
  t_ve_media_video_pixel_aspect_ratio_s     aspect_ratio;
  t_ve_media_video_stream_type_info_s       stream_type_info;
  t_ve_media_video_cyclic_intra_refresh_s   cir;
  t_ve_media_video_pid_rate_control_s       pid_rate_cntrl;
  t_ve_media_video_config_state_e           gov_header_state;
  t_ve_media_video_intra_control_s          intra_control;
  t_ve_media_video_compression_params_s     compression_params;
  UWord16                                   time_resolution;
  t_ve_media_video_frame_type_selection_s   frame_type_select;
  t_ve_media_video_encoding_mode_e          enc_mode;
  t_ve_media_video_stream_type_e            stream_type;
  t_ve_media_video_mpeg4_stream_container_s stream_container;
}t_ve_media_video_encoder_mpeg4_algorithm_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
quant_param	An enumerated type, **t_ve_media_video_quantization_param_e** as described in Appendix B.116 on Page 544, that determines the Quantization Parameter (QP) to be used during the encoding sequence.
hec_state	An enumerated type, **t_ve_media_video_config_state_e** as described in Appendix B.117 on Page 546, that enables/disables the insertion of a Header Extension Code (HEC).
video_seq_header	A structure, **t_ve_media_video_user_data_s** as described in Appendix B.29.16.1 on Page 328, that allows user data to be inserted into the Video Object Sequence (VOS) header.
video_object	A structure, **t_ve_media_video_user_data_s** as described in Appendix B.29.16.1 on Page 328, that allows user data to be inserted in the Video Object (VO) header.
video_object_layer	A structure, **t_ve_media_video_user_data_s** as described in Appendix B.29.16.1 on Page 328, that allows user data to be inserted in the Video Object Layer (VOL) header.
group_of_vop	A structure, **t_ve_media_video_user_data_s** as described in Appendix B.29.16.1 on Page 328, that allows user data to be inserted into the Group of VOP (GOV) header.
aspect_ratio	A structure, **t_ve_media_video_pixel_aspect_ratio_s** as described in Appendix B.29.16.2 on Page 329, that allows the pixel aspect ratio to be altered.
stream_type_info	A structure, **t_ve_media_video_stream_type_info_s** as described in Appendix B.29.16.3 on Page 330, that contains the MPEG-4 stream type information.
cir	A structure, **t_ve_media_video_cyclic_intra_refresh_s** as described in Appendix B.29.16.4 on Page 331, that allows configuration of the Cyclic Intra refresh (CIR) rate.

pid_rate_cntrl	A structure, **t_ve_media_video_pid_rate_control_s** as described in Appendix B.29.16.5 on Page 332, that allows configuration and control of the rate control parameters.
gov_header_state	An enumerated type, **t_ve_media_video_config_state_e** as described in Appendix B.117 on Page 546, that enables/disables the Group of VOP (GOV) header insertion into the MPEG-4 stream.
intra_control	A structure, **t_ve_media_video_intra_control_s** as described in Appendix B.29.16.7 on Page 334, that describes the penalty values for macroblock encoding type decision.
compression_params	A structure, **t_ve_media_video_compression_params_s** as described in Appendix B.29.16.8 on Page 335, that is used for accessing the MPEG-4 compression parameters.
time_resolution	Integer value that defines how many evenly spaced units (ticks) one second contains.
frame_type_select	A structure, **t_ve_media_video_frame_type_selection_s** as described in Appendix B.29.18 on Page 336, that defines the frame type to be used.
enc_mode	An enumerated type, **t_ve_media_video_encoding_mode_e** as described in Appendix B.113 on Page 542, that determines the MPEG-4 encoder type to be used.
stream_type	An enumerated type, **t_ve_media_video_stream_type_e** as described in Appendix B.120 on Page 549, that defines the MPEG-4 stream type to be used.
stream_container	A structure, **t_ve_media_video_mpeg4_stream_container_s** as described in Appendix B.29.24 on Page 342, which defines the configuration ASCII string for the MPEG-4 complete header.

B.29.16.1 t_ve_media_video_user_data_s

The **t_ve_media_video_user_data_s** structure contains user configuration parameters used in the configuration of the VeriCall Edge MPEG-4 video encoder. The **t_ve_media_video_user_data_s** structure is used within the **t_ve_media_video_encoder_mpeg4_algorithm_configuration_s** structure described in Appendix B.29.16 on Page 327, and allows user data to be inserted into the stream. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  t_ve_media_video_config_state_e   user_data_active_state;
  UWord8                pu8_user_data[VE_MEDIA_MAX_USER_DATA_SIZE];
  UWord32               u32_user_data_length;
}t_ve_media_video_user_data_s;
```

The elements of this structure are as follows:

Element	Definition
user_data_active_state	An enumerated type, **t_ve_media_video_config_state_e** as described in Appendix B.117 on Page 546, that is a simple flag that inserts or does not insert user data into the MPEG-4 stream.
pu8_user_data	Character array of the user data to be inserted into the stream. This character array must not exceed `VE_MEDIA_MAX_USER_DATA_SIZE` in length, including the termination character.
u32_user_data_length	The number of bytes to be inserted into the stream.

B.29.16.2 t_ve_media_video_pixel_aspect_ratio_s

The **t_ve_media_video_pixel_aspect_ratio_s** structure contains user configuration parameters used in the configuration of the VeriCall Edge MPEG-4 video encoder. The **t_ve_media_video_pixel_aspect_ratio_s** structure is used within the **t_ve_media_video_encoder_mpeg4_algorithm_configuration_s** structure described in Appendix B.29.16 on Page 327, and allows the pixel-aspect-ratio to be changed. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  UWord8     u8_AspectRatioInfo;
  UWord8     u8_ParWidth;
  UWord8     u8_ParHeight;
}t_ve_media_video_pixel_aspect_ratio_s;
```

The elements of this structure are as follows:

Element	Definition
u8_AspectRatioInfo	This is the pixel-aspect-ratio information as specified in the MPEG-4 standard.
u8_ParWidth	The pixel width. See the MPEG-4 standard for valid values.
u8_ParHeight	The pixel height. See the MPEG-4 standard for valid values.

B.29.16.3 t_ve_media_video_stream_type_info_s

The **t_ve_media_video_stream_type_info_s** structure contains user configuration parameters used in the configuration of the VeriCall Edge MPEG-4 video encoder. The **t_ve_media_video_stream_type_info_s** structure is used within the **t_ve_media_video_encoder_mpeg4_algorithm_configuration_s** structure described in Appendix B.29.16 on Page 327. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  t_ve_media_video_simple_profile_e    profile_and_level;
  UWord8                               u8_video_signal_type;
  UWord8                               u8_video_format;
  UWord8                               u8_video_range;
  UWord8                               u8_color_description;
  UWord8                               u8_color_primaries;
  UWord8                               u8_transfer_characteristics;
  UWord8                               u8_matrix_coefficients;
  t_ve_media_video_config_state_e      random_accessible_vol;
}t_ve_media_video_stream_type_info_s;
```

The elements of this structure are as follows:

Element	Definition
profile_and_level	An enumerated type, **t_ve_media_video_simple_profile_e** as described in Appendix B.118 on Page 547, that determines the simple MPEG-4 profile to be used.
u8_video_signal_type	The type of video signal to be used. See the MPEG-4 standard for valid values.
u8_video_format	The video format. See the MPEG-4 standard for valid values.
u8_video_range	The video range. See the MPEG-4 standard for valid values.
u8_color_description	Part of the color description information. See the MPEG-4 standard for valid values.
u8_color_primaries	Part of the color description information. See the MPEG-4 standard for valid values.
u8_transfer_characteristics	Part of the color description information. See the MPEG-4 standard for valid values.
u8_matrix_coefficients	Part of the color description information. See the MPEG-4 standard for valid values.
random_accessible_vol	An enumerated type, **t_ve_media_video_config_state_e** as described in Appendix B.117 on Page 546, that enables Random Access Video Object Layer, meaning a VOL has no dependency between video frames (all VOPs are encoded as Intra VOPs).

B.29.16.4 t_ve_media_video_cyclic_intra_refresh_s

The **t_ve_media_video_cyclic_intra_refresh_s** structure contains user configuration parameters used in the configuration of the VeriCall Edge MPEG-4 video encoder and media type. The **t_ve_media_video_cyclic_intra_refresh_s** structure is used within the **t_ve_media_video_encoder_mpeg4_algorithm_configuration_s** structure, described in Appendix B.29.16 on Page 327, and the **t_ve_media_mpeg4_configuration_s** structure, described in Appendix B.28.8 on Page 314. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  t_ve_media_video_config_state_e   cir_state;
  UWord32                           i_per_vop_percent;
}t_ve_media_video_cyclic_intra_refresh_s;
```

The elements of this structure are as follows:

Element	Definition
cir_state	An enumerated type, **t_ve_media_video_config_state_e** as described in Appendix B.117 on Page 546, that enables or disables the Cyclic Intra Refresh (CIR) .
i_per_vop_percent	The percentage of frames transmitted that will be I-Frames.

B.29.16.5 t_ve_media_video_pid_rate_control_s

The **t_ve_media_video_pid_rate_control_s** structure contains user configuration parameters used in the configuration of the VeriCall Edge MPEG-4 video encoder and media type. The **t_ve_media_video_pid_rate_control_s** structure is used within the **t_ve_media_video_encoder_mpeg4_algorithm_configuration_s** structure, described in Appendix B.29.16 on Page 327, and the **t_ve_media_mpeg4_configuration_s** structure, described in Appendix B.28.8 on Page 314, allows control of the MPEG-4 encoder rate control. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  t_ve_media_video_config_state_e           rate_ctrl_active_state;
  UWord32                                   u32_target_bit_rate;
  UWord32                                   u32_target_frame_rate;
  t_ve_media_video_config_state_e           frame_skip_state;
  t_ve_media_video_config_state_e           four_motion_vect_state;
  t_ve_media_video_quantization_param_e     default_quant_param;
  t_ve_media_video_buffer_verifier_params_s vbv_params;
}t_ve_media_video_pid_rate_control_s;
```

The elements of this structure are as follows:

Element	Definition
rate_ctrl_active_state	An enumerated type, **t_ve_media_video_config_state_e** as described in Appendix B.117 on Page 546, that enables or disables the MPEG-4 rate control.
u32_target_bit_rate	The target bit rate value. See the MPEG-4 standard for valid values.
u32_target_frame_rate	The target frames per second rate. Valid integer values are 1 to 30, any value other than this will be ignored.
frame_skip_state	An enumerated type, **t_ve_media_video_config_state_e** as described in Appendix B.117 on Page 546, that enables or disables frame skipping. If enabled the encoder can skip frames if too much data was generated by the previous frames.
four_motion_vect_state	An enumerated type, **t_ve_media_video_config_state_e** as described in Appendix B.117 on Page 546, that enables or disables the use of the four motion vectors per macroblock feature.
default_quant_param	An enumerated type, **t_ve_media_video_quantization_param_e** as described in Appendix B.116 on Page 544, that allows the encoding quantization parameter to be set.
vbv_params	A structure, **t_ve_media_video_buffer_verifier_params_s** as described in Appendix B.29.16.6 on Page 333, that is used to control the bit rate. See the Video Buffer Verifier (VBV) in the MPEG-4 standard for more information.

B.29.16.6 t_ve_media_video_buffer_verifier_params_s

The **t_ve_media_video_buffer_verifier_params_s** structure is used within the **t_ve_media_video_encoder_mpeg4_algorithm_configuration_s** structure described in Appendix B.29.16 on Page 327 and allows configuration of the MPEG-4 Video Buffer Verifier (VBV). The **t_ve_media_video_buffer_verifier_params_s** is defined in **ve_media.h** and described below:

```
typedef struct{
  UWord32    u32_set;
  UWord32    u32_bufSize;
  UWord32    u32_init_occupancy;
}t_ve_media_video_buffer_verifier_params_s;
```

The elements of this structure are as follows:

Element	Definition
u32_set	If this value is zero, the default VBV values are used as per the MPEG-4 standard. Non zero values will use the settings defined by `u32_bufSize` and `u32_init_occupancy`.
u32_bufSize	The buffer size to be used. See the MPEG-4 standard for valid values.
u32_init_occupancy	The initial VBV buffer occupancy. See the MPEG-4 standard for valid values.

B.29.16.7 t_ve_media_video_intra_control_s

The **t_ve_media_video_intra_control_s** structure contains user configuration parameters used in the configuration of the VeriCall Edge MPEG-4 video encoder. The **t_ve_media_video_intra_control_s** structure is used within the **t_ve_media_video_encoder_mpeg4_algorithm_configuration_s** structure described in Appendix B.29.16 on Page 327 and describes the penalty values for the macroblock encoding type decision. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  UWord16    u16_mv_sadpenalty;
  UWord16    u16_intra_sad_penalty;
  UWord16    u16_inter4mv_sad_penalty;
}t_ve_media_video_intra_control_s;
```

The elements of this structure are as follows:

Element	Definition
u16_mv_sadpenalty	Penalty added to none zero Motion Vector (MV) Sum of Absolute Differences (SAD).
u16_intra_sad_penalty	Penalty added to the intra SAD.
u16_inter4mv_sad_penalty	Penalty added to the four motion vector SAD.

It is recommended that these values are not changed from the default entries.

B.29.16.8 t_ve_media_video_compression_params_s

The **t_ve_media_video_compression_params_s** structure contains user configuration parameters used in the configuration of the VeriCall Edge MPEG-4 video encoder compression parameters. The **t_ve_media_video_compression_params_s** structure is used within the **t_ve_media_video_encoder_mpeg4_algorithm_configuration_s** structure described in Appendix B.29.16 on Page 327. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  t_ve_media_video_config_state_e   ac_prediction_state;
  t_ve_media_video_config_state_e   outside_vop_mot_vect_state;
  t_ve_media_video_config_state_e   rounding_ctrl_state;
  UWord8                            intra_dc_vlc_threshold;
}t_ve_media_video_compression_params_s;
```

The elements of this structure are as follows:

Element	Definition
ac_prediction_state	An enumerated type, **t_ve_media_video_config_state_e** as described in Appendix B.117 on Page 546, that enables or disables AC prediction.
outside_vop_mot_vect_state	An enumerated type, **t_ve_media_video_config_state_e** as described in Appendix B.117 on Page 546, that enables or disables motion vectors pointing outside the picture area.
rounding_ctrl_state	An enumerated type, **t_ve_media_video_config_state_e** as described in Appendix B.117 on Page 546, that enables or disables rounding control for the interpolation.
intra_dc_vlc_threshold	A threshold value defining when intra macroblock DC components should be coded with AC VLC instead of DC VLC. See the MPEG-4 standard for valid values.

B.29.17 t_ve_media_video_decoder_mpeg4_algorithm_configuration_s

The **t_ve_media_video_decoder_mpeg4_algorithm_configuration_s** structure contains configuration parameters for the VeriCall Edge H.263 video decoder and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  UWord32      reserved;
  t_ve_media_video_mpeg4_stream_container_s    stream_container;
}t_ve_media_video_decoder_mpeg4_algorithm_configuration_s;
```

The element of this structure is as follows:

Element	Definition
reserved	Reserved for future use.
stream_container	A structure, **t_ve_media_video_mpeg4_stream_container_s** as described in Appendix B.29.24 on Page 342, which defines the configuration ASCII string for the MPEG-4 complete header.

B.29.18 t_ve_media_video_frame_type_selection_s

The **t_ve_media_video_frame_type_selection_s** structure is part of the **t_ve_media_video_encoder_h263_algorithm_configuration_s** structure, described in Appendix B.29.14 on Page 325, as well as the **t_ve_media_video_encoder_mpeg4_algorithm_configuration_s** structure, described in Appendix B.29.16 on Page 327. The **t_ve_media_video_frame_type_selection_s** structure determines the video framing type to be used for the encoder. The structure is defined in **ve_media.h** and is described below:

```
typedef{
  t_ve_media_video_frame_type_selection_e    ve_frame_type;
  UWord32                                    i_frame_freq;
}t_ve_media_video_frame_type_selection_s;
```

The enumeration elements are as follows:

Element	Definition
ve_frame_type	Indicates the framing type to be used and is an enumerated type of **t_ve_media_video_frame_type_selection_e** as described in Appendix B.112 on Page 542.
i_frame_freq	Used when `ve_frame_type` is set to be both "I-Frame" and "P-Frame" mode. This element defines the frequency at which "I-Frames" are sent to the destination. For example, if `ve_frame_type` were set to be 10, then every tenth frame sent would be an "I-Frame".

B.29.19 t_ve_media_eg711_encoder_algorithm_configuration_s

The **t_ve_media_eg711_encoder_algorithm_configuration_s** structure contains configuration parameters for a GIPS enhanced G.711 encoder algorithm task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  t_ve_media_eg711_codec_and_law_e     codec_and_law;
}t_ve_media_eg711_encoder_algorithm_configuration_s;
```

The element of this structure is described below:

Element	Definition
codec_and_law	Enumerated type, **t_ve_media_eg711_codec_and_law_e**, to determine the algorithm and law to be used. The enumerated type is defined in B.111 on Page 541.

The Global IP Sound (GIPS) enhanced G.711 algorithm is not part of the default VeriCall Edge configuration. Please contact Trinity Convergence if you require more information about including GIPS within the VeriCall Edge software.

B.29.20 t_ve_media_eg711_decoder_algorithm_configuration_s

The **t_ve_media_eg711_decoder_algorithm_configuration_s** structure contains configuration parameters for a GIPS enhanced GH.711 decoder algorithm task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct{
  t_ve_media_eg711_codec_and_law_e     codec_and_law;
  UWord32                              payloadtype;
}t_ve_media_eg711_decoder_algorithm_configuration_s;
```

The elements of this structure are described below:

Element	Definition
codec_and_law	Enumerated type, **t_ve_media_eg711_codec_and_law_e**, to determine the algorithm and law to be used. The enumerated type is defined in B.111 on Page 541.
payloadtype	The specified payload type to be used with the GIPS enhanced G.711 decoder.

The Global IP Sound (GIPS) enhanced G.711 algorithm is not part of the default VeriCall Edge configuration. Please contact Trinity Convergence if you require more information about including GIPS within the VeriCall Edge software.

B.29.21 t_ve_media_ilbc_encoder_algorithm_configuration_s

The **t_ve_media_ilbc_encoder_algorithm_configuration_s** structure contains configuration parameters for an iLBC encoder algorithm task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct
{
    UWord32 frame_size;
}t_ve_media_ilbc_encoder_algorithm_configuration_s;
```

The element of this structure is as follows:

Element	Definition
frame_size	The packet interval in milliseconds. Must be either 20ms or 30ms.

B.29.22 t_ve_media_ilbc_decoder_algorithm_configuration_s

The **t_ve_media_ilbc_decoder_algorithm_configuration_s** structure contains configuration parameters for a iLBC decoder algorithm task and is part of the **t_ve_media_definition_s** structure defined Appendix B.29 on Page 315. The structure is defined in **ve_media.h** and described below:

```
typedef struct
{
    UWord32 frame_size;
}t_ve_media_ilbc_decoder_algorithm_configuration_s;
```

The element of this structure is as follows:

Element	Definition
frame_size	The packet interval in milliseconds. Must be either 20ms or 30ms.

B.29.23 t_ve_media_video_encoding_mode_rate_s

The **t_ve_media_video_encoding_mode_rate_s** structure is used within the **t_ve_media_h263_configuration_s** structure, as described in Appendix B.28.7 on Page 313, as well as within the **t_ve_media_mpeg4_configuration_s** structure, described in Appendix B.28.8 on Page 314. The **t_ve_media_video_encoding_mode_rate_s** structure defines MPEG-4 and H.263 video I-frame encoding rate. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_media_video_encoding_mode_rate_e   type;
  UWord32                                 rate;
}t_ve_media_video_encoding_mode_rate_s;
```

The elements of this structure are as follows:

Element	Definition
type	An enumerated type, **t_ve_media_video_encoding_mode_rate_e** as defined on in Appendix B.115 on Page 543, which determines the H.263 video-encoding mode rate of operation.
rate	This element defines the frequency at which “I-Frames” are sent to the destination. For example, if set to be 10, then every tenth frame sent would be an "I-Frame".

B.29.24 t_ve_media_video_mpeg4_stream_container_s

The **t_ve_media_video_mpeg4_stream_container_s** structure is used within the following structures:

- **t_ve_media_mpeg4_configuration_s** as described in Appendix B.28.8 on Page 314.
- **t_ve_media_video_encoder_mpeg4_algorithm_configuration_s** as described in Appendix B.29.16 on Page 327.
- **t_ve_media_video_decoder_mpeg4_algorithm_configuration_s** as described in Appendix B.29.17 on Page 336.

The **t_ve_media_video_mpeg4_stream_container_s** structure contains the configuration string of the MPEG-4 header. This structure is defined in **ve_media.h** and is described below:

```
typedef struct{
  Word8           mpeg4_stream_container_config_string
                                [C_MAX_MPEG4_CONFIG_STRING_LEN];
  UWord16         mpeg4_stream_container_config_string_len;
}t_ve_media_video_mpeg4_stream_container_s;
```

The elements of this structure are as follows:

Element	Definition
mpeg4_stream_container_config_string	An ASCII representation of the hexadecimal values of the complete MPEG-4 header. This string must not exceed `C_MAX_MPEG4_CONFIG_STRING_LEN` in length.
mpeg4_stream_container_config_string_len	The length of the MPEG-4 configuration string.

B.30 t_ve_network_control_u

The **t_ve_network_control_u** union is used by **VE_ControlNetwork** to over-ride the default network settings of the VeriCall Edge system. The **t_ve_network_control_u** is defined in **ve_control_network.h** and is described below:

```
typedef struct{
  UWord8     ve_tos;
}t_ve_network_control_u;
```

The union element is as follows:

Element	Definition
ve_tos	The Type Of Service (TOS) to be used. The TOS can be as per the IP TOS (RFC0791 and RFC1349) or Differentiated Services TOS (RFC2474).

B.31 t_ve_audio_control_u

The **t_ve_audio_control_u** union of structures is used in calls to the **VE_ControlAudio** function call. The encapsulated structures provide a file descriptor for the playback and recording of media data. This union and its encapsulated structures are defined in **ve_control_audio.h** and are described below:

```
typedef union{
  t_ve_audio_control_play_start_s           ve_audio_play_start;
  t_ve_audio_control_record_start_s         ve_audio_record_start;
  t_ve_audio_control_lip_sync_adjustment_s  ve_lip_sync;
  t_ve_audio_control_data_end_point_selector_s
                                            ve_data_point_select;
  t_ve_audio_control_data_audio_mute_s      ve_audio_mute;
}t_ve_audio_control_u;
```

The encapsulated structures of the **t_ve_audio_control_u** union are as follows:

Element	Definition
ve_audio_play_start	A structure, **t_ve_audio_control_play_start_s** as defined in Appendix B.31.1 on Page 344, which allows an audio file to be played into the audio stream.
ve_audio_record_start	A structure, **t_ve_audio_control_record_start_s** as defined in Appendix B.31.2 on Page 344, which allows an audio file to be recorded from the audio stream
ve_lip_sync	A structure, **t_ve_audio_control_lip_sync_adjustment_s** as defined in Appendix B.31.3 on Page 345, which controls the audio channel lip synchronization with an associated video channel.
ve_audio_mute	An enumerated type, **t_ve_audio_control_data_audio_mute_s** as described in Appendix B.31.5 on Page 346, that enables or disables frame skipping. If enabled the encoder can skip frames if too much data was generated by the previous frames.
ve_data_point_select	A structure, **t_ve_audio_control_data_end_point_selector_s** as defined in Appendix B.31.4 on Page 345, which allows the selection of a defined data-end point instance to be associated with this channel.

B.31.1 t_ve_audio_control_play_start_s

The **t_ve_audio_control_play_start_s** structure is used by **t_ve_audio_control_u** as defined in Appendix B.31 on Page 343. The **t_ve_audio_control_play_start_s** structure determines the format and direction of an audio file to be played into the audio stream. The structure is defined in **ve_control_audio.h** and is described below:

```
typedef struct{
  t_ve_audio_control_play_data_format_e   ve_format;
  t_ve_audio_control_direction_e          ve_direction;
  UWord8          descriptor[VE_CONTROL_AUDIO_MAX_FILE_DESC_SIZE];
}t_ve_audio_control_play_start_s;
```

The structure elements are as follows:

Element	Definition
ve_format	An enumerated type, **t_ve_audio_control_play_data_format_e** as defined on in Appendix B.146 on Page 571, which determines the audio file-type to be played into the audio stream.
ve_direction	An enumerated type, **t_ve_audio_control_direction_e** as defined on in Appendix B.144 on Page 570, which determines the direction of the stream (received or transmitted) to be played into.
descriptor	NULL terminated path and file descriptor name that must not exceed VE_CONTROL_AUDIO_MAX_FILE_DESC_SIZE characters in length (currently defined as being 32 characters including the termination character).

B.31.2 t_ve_audio_control_record_start_s

The **t_ve_audio_control_record_start_s** structure is used by **t_ve_audio_control_u** as defined in Appendix B.31 on Page 343. The **t_ve_audio_control_record_start_s** structure determines the format of an audio file to be recorded from the audio stream. The structure is defined in **ve_control_audio.h** and is described below:

```
typedef struct{
  t_ve_audio_control_record_data_format_e   ve_format;
  UWord8          descriptor[VE_CONTROL_AUDIO_MAX_FILE_DESC_SIZE];
}t_ve_audio_control_record_start_s;
```

The structure elements are as follows:

Element	Definition
ve_format	An enumerated type, **t_ve_audio_control_record_data_format_e** as defined on in Appendix B.145 on Page 570, which determines the audio file-type to be recorded from the audio stream.
descriptor	NULL terminated path and file descriptor name that must not exceed VE_CONTROL_AUDIO_MAX_FILE_DESC_SIZE characters in length (currently defined as being 32 characters including the termination character).

B.31.3 t_ve_audio_control_lip_sync_adjustment_s

The **t_ve_audio_control_lip_sync_adjustment_s** structure is used by **t_ve_audio_control_u** as defined in Appendix B.31 on Page 343. The **t_ve_audio_control_lip_sync_adjustment_s** structure allows the lip synchronization of the audio channel to be altered to match that of an associated video channel. The structure is defined in **ve_control_audio.h** and is described below:

```
typedef struct{
  UWord32          audio_to_video_msec_adjustment;
}t_ve_audio_control_lip_sync_adjustment_s;
```

The structure element is as follows:

Element	Definition
audio_to_video_msec_adjustment	Determines the millisecond lip-synchronization adjustment to be made on this audio channel.

B.31.4 t_ve_audio_control_data_end_point_selector_s

The **t_ve_audio_control_data_end_point_selector_s** structure is used by **t_ve_audio_control_u** as defined in Appendix B.31 on Page 343. The **t_ve_audio_control_data_end_point_selector_s** structure determines the data-end point instance to be used for this audio channel configuration. The structure is defined in **ve_control_audio.h** and is described below:

```
typedef struct{
  UWord32          data_end_point_instance;
}t_ve_audio_control_data_end_point_selector_s;
```

The structure elements are as follows:

Element	Definition
data_end_point_instance	Defines the end termination point for the channel. A data-end point instance number is associated to each VE_ATTRIBUTE_TYPE_DEFAULT_DATA_END_POINT attribute entry. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this instance.

B.31.5 t_ve_audio_control_data_audio_mute_s

The **t_ve_audio_control_data_audio_mute_s** structure is used by **t_ve_audio_control_u** as defined in Appendix B.31 on Page 343. The **t_ve_audio_control_data_audio_mute_s** structure allows the audio stream to be muted. The structure is defined in **ve_control_audio.h** and is described below:

```
typedef struct{
  t_ve_audio_control_data_audio_mute_state_e      state;
  t_ve_audio_control_data_audio_mute_direction_e  direction;
}t_ve_audio_control_data_audio_mute_s;
```

The structure elements are as follows:

Element	Definition
state	An enumerated type, **t_ve_audio_control_data_audio_mute_state_e** as defined on in Appendix B.147 on Page 572, which allows the audio stream mute to be enabled or disabled.
direction	An enumerated type, **t_ve_audio_control_data_audio_mute_direction_e** as defined on in Appendix B.148 on Page 573, which allows the audio stream direction to be muted to be configured.

B.32 t_ve_video_control_u

The **t_ve_video_control_u** union of structures used in calls to the **VE_ControlVideo** function. The encapsulated structures provide control and manipulation of the local and received video images. This union and encapsulated structures are defined in **ve_control_video.h** and are described below:

```
typedef union{
  t_ve_video_control_filter_selector_e      ve_filter_selector;
  t_ve_video_control_select_display_state_e ve_display_state;
  t_ve_video_control_scale_s                ve_scale_selector;
  t_ve_video_control_rotate_e               ve_rotation_selector;
  t_ve_video_control_mirror_e               ve_mirror;
  t_ve_video_control_mute_s                 ve_mute;
  t_ve_video_control_move_and_resize_s      ve_move_and_resize;
  t_ve_video_control_image_encode_s         ve_image_encode;
  t_ve_video_control_lip_sync_adjustment_s  ve_lip_sync;
  t_ve_video_control_data_end_point_selector_s
                                            ve_data_end_point_select;
  t_ve_video_control_image_freeze_state_e   ve_image_freeze;
  t_ve_video_control_reset_pre_proc_params_s
                                            ve_reconfigure_pre_proc;
  t_ve_video_control_reset_post_proc_params_s
                                            ve_reconfigure_post_proc;
}t_ve_video_control_u;
```

The elements of the **t_ve_video_control_u** union are as follows:

Element	Definition
ve_filter_selector	An enumerated type, **t_ve_video_control_filter_selector_e**, as defined in Appendix B.150 on Page 576. This enumerated type specifies the type of filtering, if any, to be applied to the video image.
ve_display_state	An enumerated type, **t_ve_video_control_select_display_state_e**, as defined in Appendix B.151 on Page 576. This enumerated type allows the local image to be enabled or displayed.
ve_scale_selector	A structure, **t_ve_video_control_scale_s**, as defined in Appendix B.32.1 on Page 348. This structure contains horizontal and vertical components to represent fractional scaling of the video image.
ve_rotation_selector	An enumerated type, **t_ve_video_control_rotate_e**, as defined in Appendix B.152 on Page 577. This enumerated type allows the clockwise rotation of the video image.
ve_mirror	An enumerated type, **t_ve_video_control_mirror_e**, as defined in Appendix B.153 on Page 577. This enumerated type determines the plane the video image is to be mirrored in.
ve_mute	A structure, **t_ve_video_control_mute_s**, as defined in Appendix B.32.2 on Page 349. This structure contains the mode of operation for a muted video stream.
ve_move_and_resize	A structure, **t_ve_video_control_move_and_resize_s**, as defined in Appendix B.32.3 on Page 349. This structure contains parameters to alter the size and location of the video image on the selected display.

Continued >>

Element	Definition
ve_image_encode	A structure, **t_ve_video_control_image_encode_s**, as defined in Appendix B.32.4 on Page 350. This structure contains the available encoding format for the video image.
ve_lip_sync	A structure, **t_ve_video_control_lip_sync_adjustment_s**, as defined in Appendix B.32.5 on Page 350. This structure allows adjustment of lip-synchronization between associated audio and video channels.
ve_data_end_point_select	A structure, **t_ve_video_control_data_end_point_selector_s**, as defined in Appendix B.32.6 on Page 351. This structure determines the video image size and location on the specified data-end point.
ve_image_freeze	An enumerated type, **t_ve_video_control_image_freeze_state_e**, as defined in Appendix B.155 on Page 579. This enumerated type allows the received image to be frozen.
ve_reconfigure_pre_proc	A structure, **t_ve_video_control_reset_pre_proc_params_s**, as defined in Appendix B.32.7 on Page 352. This structure allows for the redefinition of the video pre-processor parameters.
ve_reconfigure_post_proc	A structure, **t_ve_video_control_reset_post_proc_params_s**, as defined in Appendix B.32.8 on Page 354. This structure allows for the redefinition of the video post-processor parameters.

A number of these encapsulated structures do not give limits to the available settings because many of these settings are hardware specific. Platform limits for these settings will be supplied in the VeriCall Edge release notes.

B.32.1 t_ve_video_control_scale_s

The **t_ve_video_control_scale_s** structure is used by **t_ve_video_control_u** as defined in Appendix B.32 on Page 347. The **t_ve_video_control_scale_s** structure determines the scale factor to be used for the horizontal and vertical controls. The structure is defined in **ve_control_video.h** and is described below:

```
typedef struct{
  UWord32      h_scale_factor_num;
  UWord32      h_scale_factor_den;
  UWord32      v_scale_factor_num;
  UWord32      v_scale_factor_den;
}t_ve_video_control_scale_s;
```

The structure elements are as follows:

Element	Definition
h_scale_factor_num	The horizontal scale factor numerator.
h_scale_factor_den	The horizontal scale factor denominator.
v_scale_factor_num	The vertical scale factor numerator.
v_scale_factor_den	The vertical scale factor denominator.

B.32.2 t_ve_video_control_mute_s

The **t_ve_video_control_mute_s** structure is used by **t_ve_video_control_u** as defined in Appendix B.32 on Page 347. The **t_ve_video_control_mute_s** structure determines the action to be taken when the video stream is muted. The structure is defined in **ve_control_video.h** and is described below:

```
typedef struct{
  t_ve_video_control_mute_mode_e     mode
  Word8                              image[VE_VIDEO_CONTROL_IMAGE_NAME];
}t_ve_video_control_mute_s;
```

The structure elements are as follows:

Element	Definition
mode	Enumeration type, **t_ve_video_control_mute_mode_e**, as defined in Appendix B.154 on Page 578, used to determine the mute mode of operation.
image	If an image is to be sent in mute mode, this path and file descriptor determines the image to be sent. The path and file descriptor must not exceed `VE_VIDEO_CONTROL_IMAGE_NAME` in length (currently defined as being 32 characters including the termination character).

B.32.3 t_ve_video_control_move_and_resize_s

The **t_ve_video_control_move_and_resize_s** structure is used by **t_ve_video_control_u** as defined in Appendix B.32 on Page 347. The **t_ve_video_control_move_and_resize_s** structure determines the size and location on the display of the video image with auto-scaling to the required size. The structure is defined in **ve_control_video.h** and is described below:

```
typedef struct{
  UWord32    image_h_origin;
  Uword32    image_v_origin;
  UWord32    image_h_size;
  UWord32    image_v_size;
}t_ve_video_control_move_and_resize_s;
```

The structure elements are as follows:

Element	Definition
image_h_origin	The absolute horizontal location of the video image.
image_v_origin	The absolute vertical location of the video image.
image_h_size	The horizontal size of the video image.
image_v_size	The vertical size of the video image.

B.32.4 t_ve_video_control_image_encode_s

The **t_ve_video_control_image_encode_s** structure is used by **t_ve_video_control_u** as defined in Appendix B.32 on Page 347. The **t_ve_video_control_image_encode_s** structure determines the encoding format to be performed on the video image. The structure is defined in **ve_control_video.h** and is described below:

```
typedef struct{
  t_ve_video_control_encode_format_e   encode_selector;
  Word8                image[VE_VIDEO_CONTROL_IMAGE_NAME];
}t_ve_video_control_image_encode_s;
```

The structure elements of **t_ve_video_control_image_encode_s** are as follows:

Element	Definition
encode_selector	An enumerated type, **t_ve_video_control_encode_format_e** as described in Appendix B.156 on Page 579, that selects the current video encode format to be used.
image	File descriptor for the resultant encode image format, which must not exceed `VE_VIDEO_CONTROL_IMAGE_NAME` in length (currently defined as being 32 characters including the termination character).

B.32.5 t_ve_video_control_lip_sync_adjustment_s

The **t_ve_video_control_lip_sync_adjustment_s** structure is used by **t_ve_video_control_u** as defined in Appendix B.32 on Page 347. The **t_ve_video_control_lip_sync_adjustment_s** structure allows the adjustment of lip-synchronization between associated audio and video channels. The structure is defined in **ve_control_video.h** and is described below:

```
typedef struct{
  UWord32    video_to_audio_msec_adjustment;
}t_ve_video_control_lip_sync_adjustment_s;
```

The structure elements are as follows:

Element	Definition
video_to_audio_msec_adjustment	Adjusts the video packet delivery by the defined millisecond value.

B.32.6 t_ve_video_control_data_end_point_selector_s

The **t_ve_video_control_data_end_point_selector_s** structure is used by **t_ve_video_control_u** as defined in Appendix B.32 on Page 347. The **t_ve_video_control_data_end_point_selector_s** structure determines the data-end point instance and the video image size and location on the specified data-end point. The structure is defined in **ve_control_video.h** and is described below:

```
typedef struct{
  UWord32      data_end_point_instance;
  UWord32      image_v_origin;
  UWord32      image_h_origin;
  UWord32      image_v_size;
  UWord32      image_h_size;
}t_ve_video_control_data_end_point_selector_s;
```

The structure elements are as follows:

Element	Definition
data_end_point_instance	Defines the end termination point for the channel. A data-end point instance number is associated to each VE_ATTRIBUTE_TYPE_DEFAULT_DATA_END_POINT attribute entry. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this instance.
image_v_origin	The absolute vertical location on the display of the video image.
image_h_origin	The absolute horizontal location on the display of the video image.
image_v_size	The vertical size of the video image.
image_h_size	The horizontal size of the video image.

B.32.7 t_ve_video_control_reset_pre_proc_params_s

The **t_ve_video_control_reset_pre_proc_params_s** structure is used by **t_ve_video_control_u** as defined in Appendix B.32 on Page 347. The **t_ve_video_control_reset_pre_proc_params_s** structure allows the video pre-processor parameters to be altered. The structure is defined in **ve_control_video.h** and is described below:

```
typedef struct{
  UWord32                       local_image_h_origin;
  UWord32                       local_image_v_origin;
  UWord32                       local_image_h_size;
  UWord32                       local_image_v_size;
  UWord32                       local_image_h_scale_factor_num;
  UWord32                       local_image_h_scale_factor_den;
  UWord32                       local_image_v_scale_factor_num;
  UWord32                       local_image_v_scale_factor_den;
  t_ve_video_control_rotate_e   local_image_rotation;
  t_ve_video_control_mirror_e   local_image_mirror;
  UWord32                       local_image_data_end_point_instance;
  UWord32                       camera_h_size;
  UWord32                       camera_v_size;
  UWord32                       camera_data_end_point_instance;
  UWord32                       tx_image_h_size;
  UWord32                       tx_image_v_size;
  UWord32                       tx_image_h_scale_factor_num;
  UWord32                       tx_image_h_scale_factor_den;
  UWord32                       tx_image_v_scale_factor_num;
  UWord32                       tx_image_v_scale_factor_den;
  t_ve_video_control_rotate_e   tx_image_rotation;
  t_ve_video_control_mirror_e   tx_image_mirror;
}t_ve_video_control_reset_pre_proc_params_s;
```

The structure elements are as follows:

Element	Definition
local_image_h_origin	Provides a relative offset to the top left hand corner of the display on the *x*-axis for the local image. The origin should be specified as an integer.
local_image_v_origin	Provides a relative offset to the top left hand corner of the display on the *y-axis* for the local image. The origin should be specified as an integer.
local_image_h_size	Specifies the width of the image that is sent from the local camera to the local display. The width should be specified as an integer.
local_image_v_size	Specifies the height of the image that is sent from the local camera to the local display. The height should be specified as an integer.
local_image_h_scale_factor_num	The horizontal scale factor numerator. Used in conjunction with local_image_h_scale_factor_den, it allows definition of the scale ratio for the horizontal width of the local image.
local_image_h_scale_factor_den	The horizontal scale factor denominator. Used in conjunction with local_image_h_scale_factor_num, it allows definition of the scale ratio for the horizontal width of the local image.

Continuation of the t_ve_video_control_reset_pre_proc_params_s structure elements:

Element	Definition
local_image_v_scale_factor_num	The vertical scale factor numerator. Used in conjunction with local_image_v_scale_factor_den, it allows definition of the scale ratio for the vertical height of the local image.
local_image_v_scale_factor_den	The vertical scale factor denominator. Used in conjunction with local_image_v_scale_factor_num, it allows definition of the scale ratio for the vertical height of the local image.
local_image_rotation	An enumerated type, **t_ve_video_control_rotate_e**, as defined in Appendix B.152 on Page 577. This enumerated type allows the clockwise rotation of the video image.
local_image_mirror	An enumerated type, **t_ve_video_control_mirror_e**, as defined in Appendix B.153 on Page 577. This enumerated type determines the plane the local video image is to be mirrored in.
local_image_data_end_point_instance	The local image data-end point instance. A data-end point instance number is associated to each VE_ATTRIBUTE_TYPE_DEFAULT_DATA_END_POINT attribute entry. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this instance.
camera_h_size	Determines the width of the image from the local camera that is written to the frame buffer (all received data is written to a frame buffer).
camera_v_size	Determines the height of the image from the local camera that is written to the frame buffer.
camera_data_end_point_instance	The camera data-end point instance. A data-end point instance number is associated to each VE_ATTRIBUTE_TYPE_DEFAULT_DATA_END_POINT attribute entry. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this instance.
tx_image_h_size	The horizontal size of the image passed through to the encoder.
tx_image_v_size	The vertical size of the image passed through to the encoder.
tx_image_h_scale_factor_num	The horizontal scale factor numerator. Used in conjunction with tx_image_h_scale_factor_den, it allows definition of the scale ratio for the horizontal width of the image passed through to the encoder.
tx_image_h_scale_factor_den	The horizontal scale factor denominator. Used in conjunction with tx_image_h_scale_factor_num, it allows definition of the scale ratio for the horizontal width of the image passed through to the encoder.
tx_image_v_scale_factor_num	The vertical scale factor numerator. Used in conjunction with tx_image_v_scale_factor_den, it allows definition of the scale ratio for the vertical height of the image passed through to the encoder.
tx_image_v_scale_factor_den	The vertical scale factor denominator. Used in conjunction with tx_image_v_scale_factor_num, it allows definition of the scale ratio for the vertical height of the image passed through to the encoder.
tx_image_rotation	An enumerated type, **t_ve_video_control_rotate_e**, as defined in Appendix B.152 on Page 577. This enumerated type allows the clockwise rotation of the video image.
tx_image_mirror	An enumerated type, **t_ve_video_control_mirror_e**, as defined in Appendix B.153 on Page 577. This enumerated type determines the plane the video image to be transmitted is to be mirrored in.

B.32.8 t_ve_video_control_reset_post_proc_params_s

The **t_ve_video_control_reset_post_proc_params_s** structure is used by **t_ve_video_control_u** as defined in Appendix B.32 on Page 347. The **t_ve_video_control_reset_post_proc_params_s** structure allows the video post-processor parameters to be altered. The structure is defined in **ve_control_video.h** and is described below:

```
typedef struct{
  UWord32                                rx_image_h_origin;
  UWord32                                rx_image_v_origin;
  UWord32                                rx_image_h_size;
  UWord32                                rx_image_v_size;
  UWord32                                rx_image_h_scale_factor_num;
  UWord32                                rx_image_h_scale_factor_den;
  UWord32                                rx_image_v_scale_factor_num;
  UWord32                                rx_image_v_scale_factor_den;
  t_ve_video_control_filter_selector_e rx_image_filter_selection;
  t_ve_video_control_rotate_e          rx_image_rotation;
  t_ve_video_control_mirror_e          rx_image_mirror;
  UWord32                          rx_image_data_end_point_instance;
}t_ve_video_control_reset_post_proc_params_s;
```

The structure elements are as follows:

Element	Definition
rx_image_h_origin	Provides a relative offset to the top left hand corner of the display on the *x*-axis for the received image. The origin should be specified as an integer.
rx_image_v_origin	Provides a relative offset to the top left hand corner of the display on the *y-axis* for the received image. The origin should be specified as an integer.
rx_image_h_size	Specifies the width of the received image to be displayed. The width should be specified as an integer.
rx_image_v_size	Specifies the height of the received image to be displayed. The height should be specified as an integer.
rx_image_h_scale_factor_num	The horizontal scale factor numerator. Used in conjunction with rx_image_h_scale_factor_den, it allows definition of the scale ratio for the horizontal width of the received image to be displayed.
rx_image_h_scale_factor_den	The horizontal scale factor denominator. Used in conjunction with rx_image_h_scale_factor_num, it allows definition of the scale ratio for the horizontal of the received image to be displayed.
rx_image_v_scale_factor_num	The vertical scale factor numerator. Used in conjunction with rx_image_v_scale_factor_den, it allows definition of the scale ratio for the vertical height of the received image to be displayed.
rx_image_v_scale_factor_den	The vertical scale factor denominator. Used in conjunction with rx_image_v_scale_factor_num, it allows definition of the scale ratio for the vertical height of the received image to be displayed.

Continued >>

Element	Definition
rx_image_filter_selection	Enumerated type, ,**t_ve_video_control_filter_selector_e** used to allow selection of de-ringing and de-blocking functionality of the received image. The enumerated type is defined in Appendix B.150 on Page 576.
rx_image_rotation	Enumerated type, **t_ve_video_control_rotate_e**, used to determine the rotation factor to be applied to the received image. The enumerated type is defined in Appendix B.152 on Page 577.
rx_image_mirror	Enumerated type, **t_ve_video_control_mirror_e**, used to determine the mirroring factor to be applied to the received image. The enumerated type is defined in Appendix B.153 on Page 577.
rx_image_data_end_point_instance	The received image data-end point instance. A data-end point instance number is associated to each VE_ATTRIBUTE_TYPE_DEFAULT_DATA_END_POINT attribute entry. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this instance.

B.33 t_ve_user_msg_s

The **t_ve_user_msg_s** structure is used in calls to the **VE_GetNextEvent** function and contains information relating to the queued event. Through this mechanism, the VeriCall Edge event handler, the user application is notified of events. This structure is defined in **ve_events.h** and is described below:

```
typedef struct{
  t_ve_user_msg_type_e       message_type;
  t_ve_user_msg_u            message;
}t_ve_user_msg_s;

typedef union{
 t_ve_user_msg_nat_discover_response_s   nat_discover_response;
 t_ve_user_msg_nat_user_addresses_response_s
                                        nat_user_addresses_response;
 t_ve_user_msg_nat_request_failed_s    nat_request_failed;
 t_ve_user_msg_nat_keep_alive_event_s  nat_keep_alive_event;

 t_ve_user_msg_sip_session_started_s       sip_session_started;
 t_ve_user_msg_sip_session_updated_s       sip_session_updated;
 t_ve_user_msg_sip_session_stopped_s       sip_session_stopped;
 t_ve_user_msg_sip_invite_rx_s             sip_invite_rx;
 t_ve_user_msg_sip_reinvite_rx_s           sip_reinvite_rx;
 t_ve_user_msg_sip_invite_response_rx_s    sip_invite_response_rx;
 t_ve_user_msg_sip_update_rx_s             sip_update_rx;
 t_ve_user_msg_sip_update_response_rx_s    sip_update_response_rx;
 t_ve_user_msg_sip_reinvite_response_rx_s sip_reinvite_response_rx;
 t_ve_user_msg_sip_register_response_rx_s sip_register_response_rx;
 t_ve_user_msg_sip_refer_rx_s              sip_refer_rx;
 t_ve_user_msg_sip_refer_response_rx_s     sip_refer_response_rx;
 t_ve_user_msg_sip_refer_notify_rx_s       sip_refer_notify_rx;
 t_ve_user_msg_sip_message_rx_s            sip_message_rx;
 t_ve_user_msg_sip_message_response_rx_s   sip_message_response_rx;
 t_ve_user_msg_sip_options_rx_s            sip_options_rx;
 t_ve_user_msg_sip_options_response_rx_s   sip_options_response_rx;
 t_ve_user_msg_sip_info_rx_s               sip_info_rx;
 t_ve_user_msg_sip_info_response_rx_s      sip_info_response_rx;
 t_ve_user_msg_sip_service_rx_s            sip_service_rx;
 t_ve_user_msg_sip_service_response_rx_s   sip_service_response_rx;
 t_ve_user_msg_sip_operation_failed_s      sip_operation_failed;
 t_ve_user_msg_sip_presence_subscribe_rx_s
                                       sip_presence_subscribe_rx;
 t_ve_user_msg_sip_presence_notify_rx_s    sip_presence_notify_rx;
 t_ve_user_msg_sip_presence_subscribe_response_rx_s
                               sip_presence_subscribe_response_rx;
 t_ve_user_msg_sip_mw_notify_rx_s          sip_mw_notify_rx;
 t_ve_user_msg_sip_mw_subscribe_response_rx_s
                               sip_mw_subscribe_response_rx;
 t_ve_user_msg_sip_prack_rx_s              sip_prack_rx;
```

Continued >>

```
    t_ve_user_msg_sip_dialog_subscribe_rx_s sip_dialog_subscribe_rx;
    t_ve_user_msg_sip_dialog_notify_rx_s    sip_dialog_notify_rx;
    t_ve_user_msg_sip_dialog_subscribe_response_rx_s
                                    sip_dialog_subscribe_response_rx;
    t_ve_user_msg_h323_registration_response_s
                                        h323_registration_response;
    t_ve_user_msg_H323_call_started_s    h323_call_started;
    t_ve_user_msg_H323_call_stopped_s    h323_call_stopped;
    t_ve_user_msg_H323_call_updated_s    h323_call_updated;
    t_ve_user_msg_h323_call_request_s    h323_call_request;
    t_ve_user_msg_h323_video_feature_request_s
                                        h323_video_feature_request;
    t_ve_user_msg_h323_call_response_s        h323_call_response;
    t_ve_user_msg_h323_transfer_response_s    h323_transfer_response;
    t_ve_user_msg_h323_call_forwarded_s       h323_call_forwarded;
    t_ve_user_msg_h323_forward_response_s     h323_forward_response;
    t_ve_user_msg_h323_user_input_received_s
                                        h323_user_input_received;
    t_ve_user_msg_h323_call_hold_response_s
                                        h323_call_hold_response;
    t_ve_user_msg_h323_mwi_response_s         h323_mwi_response;
    t_ve_user_msg_h323_message_waiting_s      h323_message_waiting;
    t_ve_user_msg_h323_request_failed_s       h323_request_failed;

    t_ve_user_msg_detect_tone_dtmf_s          detect_dtmf_tone;
    t_ve_user_msg_detect_tone_cp_s            detect_cp_tone;
    t_ve_user_msg_detect_tone_2100_s          detect_2100_tone;
    t_ve_user_msg_detect_tone_1100_s          detect_1100_tone;
    t_ve_user_msg_received_rfc2833_digit_s    received_rfc2833_digit;
    t_ve_user_msg_play_prompt_event_s         play_prompt_event;
    t_ve_user_msg_record_prompt_event_s       record_prompt_event;
    t_ve_user_msg_encode_image_event_s        encode_image_event;
    t_ve_user_msg_video_encode_path_config_err_s
                                          video_enc_config_error;
    t_ve_user_msg_stream_event_s          stream_event;
    t_ve_user_msg_module_state_transition_event_s
                                          state_transition_error;
    t_ve_user_msg_alarm_s                 alarm;
    t_ve_user_msg_shutdown_complete_s     shutdown_complete;
    t_ve_user_msg_network_connection_event_s
                                          network_connection_event;

    t_ve_user_msg_undecodeable_stream_s   undecodable_stream;
    t_ve_user_msg_net_lookup_response_s   net_lookup_response;
    t_ve_user_msg_net_request_failed_s    net_request_failed;
    t_ve_user_msg_decode_media_change_event_s
                                          decode_media_change;
}t_ve_user_msg_u;
```

Continued >>

The elements of the **t_ve_user_msg_s** structure are as follows:

Element	Definition
message_ type	The enumerated type of message. The enumeration **t_ve_user_msg_type_e** as described in Appendix B.162 on Page 584.
message	A union of one of the following structures: **t_ve_user_msg_nat_discover_response_s** as described in Appendix B.33.1on Page 360. **t_ve_user_msg_nat_user_addresses_response_s** as described in Appendix B.33.2 on Page 360. **t_ve_user_msg_nat_request_failed_s** as described in Appendix B.33.3 on Page 361. **t_ve_user_msg_nat_keep_alive_event_s** as described in Appendix B.33.4on Page 362. **t_ve_user_msg_sip_session_started_s** as described in Appendix B.33.5 on Page 363. **t_ve_user_msg_sip_session_updated_s** as described in Appendix B.33.6 on Page 364. **t_ve_user_msg_sip_session_stopped_s** as described in Appendix B.33.7 on Page 364. **t_ve_user_msg_sip_invite_rx_s** as described in Appendix B.33.8 on Page 365. **t_ve_user_msg_sip_reinvite_rx_s** as described in Appendix B.33.9 on Page 366. **t_ve_user_msg_sip_invite_response_rx_s** as described in Appendix B.33.10 on Page 367. **t_ve_user_msg_sip_update_rx_s** as described in Appendix B.33.11 on Page 368. **t_ve_user_msg_sip_update_response_rx_s** as described in Appendix B.33.12 on Page 369. **t_ve_user_msg_sip_reinvite_response_rx_s** as described in Appendix B.33.13 on Page 370. **t_ve_user_msg_sip_register_response_rx_s** as described in Appendix B.33.14 on Page 371. **t_ve_user_msg_sip_refer_rx_s** as described in Appendix B.33.15 on Page 372. **t_ve_user_msg_sip_refer_response_rx_s** as described in Appendix B.33.16 on Page 373. **t_ve_user_msg_sip_refer_notify_rx_s** as described in Appendix B.33.17 on Page 374. **t_ve_user_msg_sip_message_rx_s** as described in Appendix B.33.18 on Page 375. **t_ve_user_msg_sip_message_response_rx_s** as described in Appendix B.33.19 on Page 376. **t_ve_user_msg_sip_options_rx_s** as described in Appendix B.33.20 on Page 377. **t_ve_user_msg_sip_options_response_rx_s** as described in Appendix B.33.21 on Page 378. **t_ve_user_msg_sip_info_rx_s** as described in Appendix B.33.22 on Page 379. **t_ve_user_msg_sip_info_response_rx_s** as described in Appendix B.33.23 on Page 380. **t_ve_user_msg_sip_service_rx_s** as described in Appendix B.33.24 on Page 381. **t_ve_user_msg_sip_service_response_rx_s** as described in Appendix B.33.25 on Page 382. **t_ve_user_msg_sip_operation_failed_s** as described in Appendix B.33.26 on Page 383. **t_ve_user_msg_sip_presence_subscribe_rx_s** as described in Appendix B.33.27 on Page 384. **t_ve_user_msg_sip_presence_notify_rx_s** as described in Appendix B.33.28 on Page 385. **t_ve_user_msg_sip_presence_subscribe_response_rx_s** as described in Appendix B.33.29 on Page 386. **t_ve_user_msg_sip_mw_notify_rx_s** as described in Appendix B.33.30 on Page 387. **t_ve_user_msg_sip_mw_subscribe_response_rx_s** as described in Appendix B.33.31 on Page 388. **t_ve_user_msg_sip_prack_rx_s** as described in Appendix B.33.32 on Page 389. **t_ve_user_msg_sip_dialog_subscribe_rx_s** as described in Appendix B.33.33 on Page 391. **t_ve_user_msg_sip_dialog_notify_rx_s** as described in Appendix B.33.34 on Page 392. **t_ve_user_msg_sip_dialog_subscribe_response_rx_s** as described in Appendix B.33.35 on Page 393.

Element	Definition
	t_ve_user_msg_h323_registration_response_s as described in Appendix B.33.36 on Page 394. **t_ve_user_msg_h323_call_started_s** as described in Appendix B.33.37 on Page 395. **t_ve_user_msg_h323_call_stopped_s** as described in Appendix B.33.38 on Page 396. **t_ve_user_msg_h323_call_updated_s** as described in Appendix B.33.39 on Page 396. **t_ve_user_msg_h323_call_request_s** as described in Appendix B.33.40 on Page 397. **t_ve_user_msg_h323_call_response_s** as described in Appendix B.33.41 on Page 397. **t_ve_user_msg_h323_transfer_response_s** as described in Appendix B.33.42 on Page 398. **t_ve_user_msg_h323_call_forwarded_s** as described in Appendix B.33.43 on Page 399. **t_ve_user_msg_h323_forward_response_s** as described in Appendix B.33.44 on Page 399. **t_ve_user_msg_h323_user_input_received_s** as described in Appendix B.33.45 on Page 400. **t_ve_user_msg_h323_call_hold_response_s** as described in Appendix B.33.46 on Page 401. **t_ve_user_msg_h323_mwi_response_s** as described in Appendix B.33.47 on Page 401. **t_ve_user_msg_h323_message_waiting_s** as described in Appendix B.33.48 on Page 402. **t_ve_user_msg_h323_request_failed_s** as described in Appendix B.33.49 on Page 403. **t_ve_user_msg_h323_video_feature_request_s** as described in Appendix B.33.50 on Page 404. **t_ve_user_msg_detect_tone_dtmf_s** as described in Appendix B.33.51 on Page 404. **t_ve_user_msg_detect_tone_cp_s** as described in Appendix B.33.52 on Page 405. **t_ve_user_msg_detect_tone_2100_s** as described in Appendix B.33.53 on Page 405. **t_ve_user_msg_detect_tone_1100_s** as described in Appendix B.33.54 on Page 406. **t_ve_user_msg_received_rfc2833_digit_s** as described in Appendix B.33.55 on Page 406. **t_ve_user_msg_play_prompt_event_s** as described in Appendix B.33.56 on Page 407. **t_ve_user_msg_record_prompt_event_s** as described in Appendix B.33.57 on Page 407. **t_ve_user_msg_encode_image_event_s** as described in Appendix B.33.58 on Page 408. **t_ve_user_msg_video_encode_path_config_err_s** as described in Appendix B.33.59 on Page 409. **t_ve_user_msg_stream_event_s** as described in Appendix B.33.60 on Page 411. **t_ve_user_msg_module_state_transition_event_s** as described in Appendix B.33.61 on Page 412. **t_ve_user_msg_alarm_s** as described in Appendix B.33.62 on Page 413. **t_ve_user_msg_shutdown_complete_s** as described in Appendix B.33.63 on Page 417. **t_ve_user_msg_network_connection_event_s** as described in Appendix B.33.64 on Page 417. **t_ve_user_msg_undecodeable_stream_s** as described in Appendix B.33.67 on Page 422. **t_ve_user_msg_net_lookup_response_s** as described in Appendix B.33.65 on Page 418. **t_ve_user_msg_net_request_failed_s** as described in Appendix B.33.66 on Page 421. **t_ve_user_msg_decode_media_change_event_s** as described in Appendix B.33.68 on Page 423.

B.33.1 t_ve_user_msg_nat_discover_response_s

The **t_ve_user_msg_nat_discover_response_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The structure contains the NAT traversal module configuration parameters and sub-module states. This structure is defined in **ve_nat_events.h** and is described below.

```
typedef struct{
  t_ve_nat_configuration_s     config;
}t_ve_user_msg_nat_discover_response_s;
```

The element of this structure is as follows:

Element	Definition
config	Structure of type **t_ve_nat_configuration_s**, as described in Appendix B.63 on Page 496, which contains the NAT traversal module configuration details.

B.33.2 t_ve_user_msg_nat_user_addresses_response_s

The **t_ve_user_msg_nat_user_addresses_response_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The structure contains the user addresses being used for the specified user. This structure is defined in **ve_nat_events.h** and is described below.

```
typedef struct{
  UWord32                          remaining;
  t_ve_user_handle_s               user_handle;
  UWord32                          channel_number;
  t_ve_user_sub_channel_select_e   sub_channel;
  t_ve_nat_channel_addresses_s     chan_addr;
}t_ve_user_msg_nat_user_addresses_response_s;
```

The elements of this structure are as follows:

Element	Definition
remaining	The number of channel mappings remaining for this user.
user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
channel_number	The channel number that the event was raised on.
sub_channel	Enumerated type, **t_ve_user_sub_channel_select_e** as described in Appendix B.167 on Page 593, which specifies the sub-channel type the response is associated with.
chan_addr	Structure of type **t_ve_nat_channel_addresses_s**, as described in Appendix B.61 on Page 492, used to determine the number of NAT channel address mappings.

B.33.3 t_ve_user_msg_nat_request_failed_s

The **t_ve_user_msg_nat_request_failed_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The structure contains the response indicators for a failed NAT traversal module request. This structure is defined in **ve_nat_events.h** and is described below.

```
typedef struct{
  t_ve_nat_request_type_e              request_type;
  t_ve_nat_request_failed_reason_e     reason;
  t_ve_user_handle_s                   user_handle;
  UWord32                              channel_number;
}t_ve_user_msg_nat_request_failed_s;
```

The elements of this structure are as follows:

Element	Definition
request_type	Enumerated type, **t_ve_nat_request_type_e** as described in Appendix B.230 on Page 653, which specifies the NAT traversal module request type that has failed.
reason	Enumerated type, **t_ve_nat_request_failed_reason_e** as described in Appendix B.231 on Page 654, which specifies the reason for the failure.
user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration, if appropriate.
channel_number	The channel number that the event was raised on, if appropriate.

B.33.4 t_ve_user_msg_nat_keep_alive_event_s

The **t_ve_user_msg_nat_keep_alive_event_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The structure contains information pertaining to the keep alive event from the NAT traversal module. This structure is defined in **ve_nat_events.h** and is described below.

```
typedef struct{
  t_ve_nat_keep_alive_event_e          event_type;
  t_ve_network_connection_s            network_connection;
  t_ve_media_ip_configuration_s        mapped_ip;
  UWord16                              mapped_port;
  UWord32                              duplicates;
}t_ve_user_msg_nat_keep_alive_event_s;
```

The elements of this structure are as follows:

Element	Definition
event_type	Enumerated type, **t_ve_nat_keep_alive_event_e** as described in Appendix B.232 on Page 655, which specifies the reason for this keep alive event.
network_connection	Structure of type **t_ve_network_connection_s**, as described in Appendix B.64 on Page 498, used to describe the network connection of this keep alive event.
mapped_ip	Structure of type **t_ve_media_ip_configuration_s**, as described in Appendix B.27.2 on Page 304, used to describe the mapped IP address assigned by the NAT traversal module.
mapped_port	The mapped port number assigned by the NAT traversal module.
duplicates	The number of duplicate starts.

The VeriCall Edge system uses the network byte ordering for IP addresses when a UWord32 value is required.

B.33.5 t_ve_user_msg_sip_session_started_s

The **t_ve_user_msg_sip_session_started_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The structure contains the user and session handle of the stream under consideration as well as the media configuration capabilities of the remote end. This structure is defined in **ve_sip_events.h** and is described below.

```
typedef struct{
  t_ve_user_handle_s                ve_user_handle;
  t_ve_sip_session_handle_s         ve_sip_session_handle;
  t_ve_stream_configuration_s       ve_stream_config;
}t_ve_user_msg_sip_session_started_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	The handle of the SIP session under consideration as described in Appendix B.49 on Page 463.
ve_stream_config	Structure of type **t_ve_stream_configuration_s**, as described in Appendix B.24 on Page 298, used to return the negotiated call parameters.

B.33.6 t_ve_user_msg_sip_session_updated_s

The **t_ve_user_msg_sip_session_started_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The structure contains the user and session handle of the stream under consideration as well as the media configuration capabilities of the remote end. This structure is defined in **ve_sip_events.h** and is described below.

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_session_handle;
  t_ve_stream_configuration_s     ve_stream_config;
}t_ve_user_msg_sip_session_updated_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	The handle of the SIP session under consideration as described in Appendix B.49 on Page 463.
ve_stream_config	Structure of type **t_ve_stream_configuration_s**, as described in Appendix B.24 on Page 298, used to return the negotiated call parameters.

B.33.7 t_ve_user_msg_sip_session_stopped_s

The **t_ve_user_msg_sip_session_stopped_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The **t_ve_user_msg_sip_session_stopped_s** structure provides the appropriate identifying handles of the SIP session under consideration. The **t_ve_user_msg_sip_session_stopped_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_session_handle;
}t_ve_user_msg_sip_session_stopped_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session under consideration.

B.33.8 t_ve_user_msg_sip_invite_rx_s

The **t_ve_user_msg_sip_invite_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356. The structure contains the information of the received SIP invite message. The **t_ve_user_msg_sip_invite_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_stream_configuration_s     ve_stream_config;
  t_ve_sip_session_handle_s       ve_sip_replaced_session_handle;
}t_ve_user_msg_sip_invite_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session under consideration.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
ve_stream_config	Structure of type **t_ve_stream_configuration_s**, as described in Appendix B.24 on Page 298, used to return the negotiated call parameters.
ve_sip_replaced_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to replace the existing SIP session handle.

B.33.9 t_ve_user_msg_sip_reinvite_rx_s

The **t_ve_user_msg_sip_reinvite_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356. The structure contains the information of the received SIP re-invite message. The **t_ve_user_msg_sip_reinvite_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_stream_configuration_s     ve_stream_config;
  t_ve_sip_session_handle_s       ve_sip_replaced_session_handle;
}t_ve_user_msg_sip_reinvite_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session under consideration.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
ve_stream_config	Structure of type **t_ve_stream_configuration_s**, as described in Appendix B.24 on Page 298, used to return the negotiated call parameters.
ve_sip_replaced_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to replace the existing SIP session handle.

B.33.10 t_ve_user_msg_sip_invite_response_rx_s

The **t_ve_user_msg_sip_invite_response_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and returns a standard SIP response code to a SIP INVITE message. The **t_ve_user_msg_sip_invite_response_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_sip_response_code_e        response_code;
  t_ve_sip_rx_session_response_data_s
                                  ve_sip_rx_session_response_data;
}t_ve_user_msg_sip_invite_response_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session under consideration.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
response_code	An enumerated SIP response code that will be in the range of 100 to 600. The enumerated type, **t_ve_sip_response_code_e**, is defined in Appendix B.181 on Page 602.
ve_sip_rx_session_response_data	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, that contains the received session response data.

B.33.11 t_ve_user_msg_sip_update_rx_s

The **t_ve_user_msg_sip_update_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and returns a standard user and SIP identifying handles as well as the current stream configuration. The **t_ve_user_msg_sip_update_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_call_session_handle;
  t_ve_sip_session_handle_s       ve_sip_update_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_stream_configuration_s     ve_stream_config;
}t_ve_user_msg_sip_update_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_call_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session under consideration.
ve_sip_update_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the update SIP session under consideration.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
ve_stream_config	Structure of type **t_ve_stream_configuration_s**, as described in Appendix B.24 on Page 298, used to return the negotiated call parameters.

B.33.12 t_ve_user_msg_sip_update_response_rx_s

The **t_ve_user_msg_sip_update_response_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and returns a standard SIP response code as well as the user and SIP identifying handles. The **t_ve_user_msg_sip_update_response_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_call_session_handle;
  t_ve_sip_session_handle_s       ve_sip_update_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_sip_response_code_e        response_code;
  t_ve_sip_rx_session_response_data_s
                                  ve_sip_rx_session_response_data;
}t_ve_user_msg_sip_update_response_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_call_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session under consideration.
ve_sip_update_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the updated SIP session.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
response_code	An enumerated SIP response code that will be in the range of 100 to 600. The enumerated type, **t_ve_sip_response_code_e**, is defined in Appendix B.181 on Page 602.
ve_sip_rx_session_response_data	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, that contains the received session response data.

B.33.13 t_ve_user_msg_sip_reinvite_response_rx_s

The **t_ve_user_msg_sip_reinvite_response_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and returns a standard SIP response code to a SIP REINVITE message as well as providing the identifying user and SIP session handles. The **t_ve_user_msg_sip_reinvite_response_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_sip_response_code_e        response_code;
  t_ve_sip_rx_session_response_data_s
                                  ve_sip_rx_session_response_data;
  t_ve_sip_call_hold_state_e      ve_sip_call_hold_state;
}t_ve_user_msg_sip_reinvite_response_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session under consideration.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
response_code	An enumerated SIP response code that will be in the range of 100 to 600. The enumerated type, **t_ve_sip_response_code_e**, is defined in Appendix B.181 on Page 602.
ve_sip_rx_session_response_data	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, that contains the received session response data.
ve_sip_call_hold_state	An enumerated type, **t_ve_sip_call_hold_state_e** as described in Appendix B.191 on Page 609. The enumeration defines the current call hold state.

B.33.14 t_ve_user_msg_sip_register_response_rx_s

The **t_ve_user_msg_sip_register_response_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and returns a standard SIP response code to a SIP registration request message. The **t_ve_user_msg_sip_register_response_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s                ve_user_handle;
  t_ve_sip_session_response_e       response_code;
  t_ve_sip_call_id_s                ve_sip_call_id;
}t_ve_user_msg_sip_register_response_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
response_code	An enumerated SIP response code that will be in the range of 100 to 600. The enumerated type, **t_ve_sip_response_code_e**, is defined in Appendix B.181 on Page 602.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.

B.33.15 t_ve_user_msg_sip_refer_rx_s

The **t_ve_user_msg_sip_refer_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and returns the necessary information when a SIP refer message has been received. The **t_ve_user_msg_sip_refer_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_refer_session_handle;
  t_ve_sip_session_handle_s       ve_sip_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_sip_user_uri_s             refer_to;
}t_ve_user_msg_sip_refer_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_refer_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session being referred to.
ve_sip_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the originating SIP session under consideration.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
refer_to	Structure of type **t_ve_sip_user_uri_s**, as described in Appendix B.39.1 on Page 440, used to describe the URI of the SIP session being referred to.

B.33.16 t_ve_user_msg_sip_refer_response_rx_s

The **t_ve_user_msg_sip_refer_response_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and returns the necessary information for a response to a received refer SIP message. The **t_ve_user_msg_sip_refer_response_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_refer_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_sip_response_code_e        response_code;
}t_ve_user_msg_sip_refer_response_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_refer_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the referred to SIP session.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
response_code	An enumerated SIP response code that will be in the range of 100 to 600. The enumerated type, **t_ve_sip_response_code_e**, is defined in Appendix B.181 on Page 602.

B.33.17 t_ve_user_msg_sip_refer_notify_rx_s

The **t_ve_user_msg_sip_refer_notify_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and returns the user and SIP handles, as well as a standard SIP response code, to a refer notify SIP message. The **t_ve_user_msg_sip_refer_notify_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_refer_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_sip_response_code_e        response_code;
}t_ve_user_msg_sip_refer_notify_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_refer_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the referred to SIP session.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
response_code	An enumerated SIP response code that will be in the range of 100 to 600. The enumerated type, **t_ve_sip_response_code_e**, is defined in Appendix B.181 on Page 602.

B.33.18 t_ve_user_msg_sip_message_rx_s

The **t_ve_user_msg_sip_message_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and provides the user and SIP session handles, as well as the actual message data. The **t_ve_user_msg_sip_message_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_call_session_handle;
  t_ve_sip_session_handle_s       ve_sip_message_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_sip_content_data_s         ve_sip_content_data;
}t_ve_user_msg_sip_message_rx_s
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_call_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session under consideration.
ve_sip_message_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP message session under consideration.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
ve_sip_content_data	Structure of type **t_ve_sip_content_data_s**, as described in Appendix B.45.2 on Page 450, used to hold the SIP message data.

B.33.19 t_ve_user_msg_sip_message_response_rx_s

The **t_ve_user_msg_sip_message_response_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and provides the user and SIP session handles, as well as the actual message data. The **t_ve_user_msg_sip_message_response_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_message_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_sip_content_data_s         ve_sip_content_data;
  t_ve_sip_response_code_e        response_code;
}t_ve_user_msg_sip_message_response_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_message_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP message session under consideration.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
ve_sip_content_data	Structure of type **t_ve_sip_content_data_s**, as described in Appendix B.45.2 on Page 450, used to hold the SIP message data.
response_code	An enumerated SIP response code that will be in the range of 100 to 600. The enumerated type, **t_ve_sip_response_code_e**, is defined in Appendix B.181 on Page 602.

B.33.20 t_ve_user_msg_sip_options_rx_s

The **t_ve_user_msg_sip_options_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and provides the user with details of the received OPTIONS request. The structure contains the session handle for the received OPTIONS request as well as the actual message data. The **t_ve_user_msg_sip_options_rx_s** structure is defined in **ve_sip_events.h** and described below:

```
typedef struct{
    t_ve_user_handle_s          ve_user_handle;
    t_ve_sip_session_handle_s   ve_sip_options_session_handle;
    t_ve_sip_call_id_s          ve_sip_call_id;
    t_ve_sip_content_data_s     ve_sip_content_data;
}t_ve_user_msg_sip_options_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_options_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to identify the received OPTIONS request.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
ve_sip_content_data	Structure of type **t_ve_sip_content_data_s**, as described in Appendix B.45.2 on Page 450, used to hold the SIP information data.

B.33.21 t_ve_user_msg_sip_options_response_rx_s

The **t_ve_user_msg_sip_options_response_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and provides the user with details of the received OPTIONS response. The structure contains the session handle for the received OPTIONS response as well as the actual message data and response code. The **t_ve_user_msg_sip_options_response_rx_s** structure is defined in **ve_sip_events.h** and described below:

```
typedef struct{
    t_ve_user_handle_s          ve_user_handle;
    t_ve_sip_session_handle_s   ve_sip_options_session_handle;
    t_ve_sip_call_id_s          ve_sip_call_id;
    t_ve_sip_content_data_s     ve_sip_content_data;
    t_ve_sip_response_code_e    response_code;
}t_ve_user_msg_sip_options_response_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_options_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to identify the received OPTIONS response.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
ve_sip_content_data	Structure of type **t_ve_sip_content_data_s**, as described in Appendix B.45.2 on Page 450, used to hold the SIP information data.
response_code	An enumerated SIP response code that will be in the range of 100 to 600. The enumerated type, **t_ve_sip_response_code_e**, is defined in Appendix B.181 on Page 602.

B.33.22 t_ve_user_msg_sip_info_rx_s

The **t_ve_user_msg_sip_info_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and provides the user and SIP identifying handles, as well as the actual SIP information data. The **t_ve_user_msg_sip_info_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_call_session_handle;
  t_ve_sip_session_handle_s       ve_sip_info_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_sip_content_data_s         ve_sip_content_data;
}t_ve_user_msg_sip_info_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_call_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session under consideration.
ve_sip_info_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP information session under consideration.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
ve_sip_content_data	Structure of type **t_ve_sip_content_data_s**, as described in Appendix B.45.2 on Page 450, used to hold the SIP information data.

B.33.23 t_ve_user_msg_sip_info_response_rx_s

The **t_ve_user_msg_sip_info_response_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and provides the user and SIP identifying handles, as well as the actual SIP information data. The **t_ve_user_msg_sip_info_response_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_info_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_sip_content_data_s         ve_sip_content_data;
  t_ve_sip_response_code_e        response_code;
}t_ve_user_msg_sip_info_response_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_info_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP information session under consideration.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
ve_sip_content_data	Structure of type **t_ve_sip_content_data_s**, as described in Appendix B.45.2 on Page 450, used to hold the SIP information data.
response_code	An enumerated SIP response code that will be in the range of 100 to 600. The enumerated type, **t_ve_sip_response_code_e**, is defined in Appendix B.181 on Page 602.

B.33.24 t_ve_user_msg_sip_service_rx_s

The **t_ve_user_msg_sip_service_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and provides the user with details of the received SERVICE request. The structure contains the session handle for the received SERVICE request as well as the actual message data. The **t_ve_user_msg_sip_service_rx_s** structure is defined in **ve_sip_events.h** and described below:

```
typedef struct{
    t_ve_user_handle_s          ve_user_handle;
    t_ve_sip_session_handle_s   ve_sip_service_session_handle;
    t_ve_sip_call_id_s          ve_sip_call_id;
    t_ve_sip_content_data_s     ve_sip_content_data;
}t_ve_user_msg_sip_service_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_service_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to identify the received SERVICE request.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
ve_sip_content_data	Structure of type **t_ve_sip_content_data_s**, as described in Appendix B.45.2 on Page 450, used to hold the SIP information data.

B.33.25 t_ve_user_msg_sip_service_response_rx_s

The **t_ve_user_msg_sip_service_response_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and provides the user with details of the received SERVICE response. The structure contains the session handle for the received SERVICE response as well as the actual message data and response code. The **t_ve_user_msg_sip_service_response_rx_s** structure is defined in **ve_sip_events.h** and described below:

```
typedef struct{
    t_ve_user_handle_s          ve_user_handle;
    t_ve_sip_session_handle_s   ve_sip_service_session_handle;
    t_ve_sip_call_id_s          ve_sip_call_id;
    t_ve_sip_content_data_s     ve_sip_content_data;
    t_ve_sip_response_code_e    response_code;
}t_ve_user_msg_sip_service_response_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_service_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to identify the received SERVICE response.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
ve_sip_content_data	Structure of type **t_ve_sip_content_data_s**, as described in Appendix B.45.2 on Page 450, used to hold the SIP information data.
response_code	An enumerated SIP response code that will be in the range of 100 to 600. The enumerated type, **t_ve_sip_response_code_e**, is defined in Appendix B.181 on Page 602.

B.33.26 t_ve_user_msg_sip_operation_failed_s

The **t_ve_user_msg_sip_operation_failed_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and returns a reason for the indicated SIP operation. The **t_ve_user_msg_sip_operation_failed_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s                  ve_user_handle;
  t_ve_sip_session_handle_s           ve_sip_session_handle;
  t_ve_sip_request_type_e             request_type;
  t_ve_sip_request_failed_reason_e    reason;
}t_ve_user_msg_sip_operation_failed_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session under consideration.
request_type	An enumeration type, **t_ve_sip_request_type_e** as described in Appendix B.193 on Page 611, which indicates the SIP operation that failed.
reason	An enumeration type, **t_ve_sip_request_failed_reason_e** as described in Appendix B.166 on Page 591, which provides a reason for the SIP operation failure.

B.33.27 t_ve_user_msg_sip_presence_subscribe_rx_s

The **t_ve_user_msg_sip_presence_subscribe_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and provides the user and SIP identifying handles, as well as the current SIP subscribe state. The **t_ve_user_msg_sip_presence_subscribe_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_sip_subscribe_state_e      ve_sip_subscribe_state;
}t_ve_user_msg_sip_presence_subscribe_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session under consideration.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
ve_sip_subscribe_state	An enumeration type, **t_ve_sip_subscribe_state_e** as described in Appendix B.194 on Page 614, which provides the current subscription state.

B.33.28 t_ve_user_msg_sip_presence_notify_rx_s

The **t_ve_user_msg_sip_presence_notify_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and provide the user and SIP identifying handles, as well as the presence data. The **t_ve_user_msg_sip_presence_notify_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_sip_presence_data_s        ve_sip_presence_data;
  t_ve_sip_subscribe_state_e      ve_sip_subscribe_state;
}t_ve_user_msg_sip_presence_notify_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session under consideration.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
ve_sip_presence_data	Structure of type **t_ve_sip_presence_data_s**, as described in Appendix B.46 on Page 452, used to hold the presence data.
ve_sip_subscribe_state	An enumeration type, **t_ve_sip_subscribe_state_e** as described in Appendix B.194 on Page 614, which provides the current subscription state.

B.33.29 t_ve_user_msg_sip_presence_subscribe_response_rx_s

The **t_ve_user_msg_sip_presence_subscribe_response_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and provides the user and SIP identifying handles, as well as a standard SIP response code to a SIP SUBSCRIBE message. The **t_ve_user_msg_sip_presence_subscribe_response_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_sip_response_code_e        response_code;
}t_ve_user_msg_sip_presence_subscribe_response_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session under consideration.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
response_code	An enumerated SIP response code that will be in the range of 100 to 600. The enumerated type, **t_ve_sip_response_code_e**, is defined in Appendix B.181 on Page 602.

B.33.30 t_ve_user_msg_sip_mw_notify_rx_s

The **t_ve_user_msg_sip_mw_notify_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and provides the user and SIP identifying handles, as well as the message waiting data. The **t_ve_user_msg_sip_mw_notify_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_sip_mw_data_s              ve_sip_mw_data;
  t_ve_sip_subscribe_state_e      ve_sip_subscribe_state;
}t_ve_user_msg_sip_mw_notify_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session under consideration.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
ve_sip_mw_data	Structure of type **t_ve_sip_mw_data_s**, as described in Appendix B.47 on Page 457, used to hold the message waiting data.
ve_sip_subscribe_state	An enumeration type, **t_ve_sip_subscribe_state_e** as described in Appendix B.194 on Page 614, which provides the current subscription state.

B.33.31 t_ve_user_msg_sip_mw_notify_response_rx_s

The **t_ve_user_msg_sip_mw_notify_response_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and returns a standard SIP response code to a message waiting subscribe response SIP message. The **t_ve_user_msg_sip_mw_notify_response_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
  t_ve_sip_response_code_e        response_code;
}t_ve_user_msg_sip_mw_subscribe_response_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session under consideration.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
response_code	An enumerated SIP response code that will be in the range of 100 to 600. The enumerated type, **t_ve_sip_response_code_e**, is defined in Appendix B.181 on Page 602.

B.33.32 t_ve_user_msg_sip_prack_rx_s

The **t_ve_user_msg_sip_prack_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and provides the user and SIP identifying handles of this associated PRACK response. The **t_ve_user_msg_sip_prack_rx_s** structure is defined in **ve_sip_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_sip_session_handle_s       ve_sip_session_handle;
  t_ve_sip_call_id_s              ve_sip_call_id;
}t_ve_user_msg_sip_prack_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to describe the handle of the SIP session under consideration.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.

B.33.32.1 t_ve_sip_call_id_s

The **t_ve_sip_call_id_s** structure is used by a number of the SIP event message structures. This structure provides information on the SIP call under consideration. The **t_ve_sip_call_id_s** structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
  Word8                          request_uri[VE_MAX_URI_STRING_SIZE];
  t_ve_sip_user_uri_s            sip_to_uri;
  t_ve_sip_user_uri_s            sip_from_uri;
  t_ve_sip_contact_info_s        sip_contact_info;
  Word8              subject[VE_SIP_MAX_SUBJECT_STRING_SIZE];
  Word8              allow[VE_SIP_MAX_ALLOW_STRING_SIZE];
  t_ve_sip_date_s    date;
  t_ve_sip_privacy_identity_s  privacy_identity;
  Word8              user_agent[VE_SIP_MAX_USER_AGENT_STRING_SIZE];
  Word8              server[VE_SIP_MAX_SERVER_STRING_SIZE];
  t_ve_sip_user_uri_s            referred_by;
  Word8              alert_info[VE_SIP_MAX_ALERT_INFO_STRING_SIZE];
}t_ve_sip_call_id_s;
```

The elements of this structure are as follows:

Element	Definition
request_uri	The request URI.
sip_to_uri	A structure, **t_ve_sip_user_uri_s** as defined in Appendix B.39.1 on Page 440, which defines the SIP URI of the destination of this SIP message. Note that this structure contains an optional display name element, which may be `NULL` if the remote user has not sent the information.
sip_from_uri	A structure, **t_ve_sip_user_uri_s** as defined in Appendix B.39.1 on Page 440, which defines the SIP URI of this caller. Note that this structure contains an optional display name element, which may be `NULL` if the remote user has not sent the information.
sip_contact_info	A structure, **t_ve_sip_contact_info_s,** as defined in Appendix B.39.2 on Page 441, which defines the optional contact information of this caller.
subject	The subject SIP header, if set.
allow	The allow SIP header, if set.
date	A structure, **t_ve_sip_date_s,** as defined in Appendix B.43 on Page 447, which provides the date and time of the SIP message being sent. Note that the date information held within this structure may not always be accurate or correct.
privacy_identity	Structure of type t_ve_sip_privacy_identity_s, as described in Appendix B.50 on Page 464, identifying the privacy and identity SIP headers, if set.
user_agent	The User Agent SIP header, if set.
server	The server SIP header, if set.
referred_by	The referred-by SIP header, if set.
alert_info	The Alert-Info SIP header, if set.

B.33.33 t_ve_user_msg_sip_dialog_subscribe_rx_s

The **t_ve_user_msg_sip_dialog_subscribe_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and provides the user with details of the received dialog SUBSCRIBE request. The structure contains the session handle for the received SUBSCRIBE request as well as the current SIP SUBSCRIBE state. The **t_ve_user_msg_sip_dialog_subscribe_rx_s** structure is defined in **ve_sip_events.h** and described below:

```
typedef struct{
    t_ve_user_handle_s                  ve_user_handle;
    t_ve_sip_session_handle_s           ve_sip_session_handle;
    t_ve_sip_call_id_s                  ve_sip_call_id;
    t_ve_sip_subscribe_state_e          ve_sip_subscribe_state;
}t_ve_user_msg_sip_dialog_subscribe_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to identify the received dialog SUBSCRIBE request.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
ve_sip_subscribe_state	An enumeration type, **t_ve_sip_subscribe_state_e** as described in Appendix B.194 on Page 614, which provides the current subscription state.

B.33.34 t_ve_user_msg_sip_dialog_notify_rx_s

The **t_ve_user_msg_sip_dialog_notify_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and provides the user with details of the received dialog NOTIFY request. The structure contains the session handle for the received NOTIFY request as well as the dialog data. The **t_ve_user_msg_sip_dialog_notify_rx_s** structure is defined in **ve_sip_events.h** and described below:

```
typedef struct{
    t_ve_user_handle_s                  ve_user_handle;
    t_ve_sip_session_handle_s           ve_sip_session_handle;
    t_ve_sip_call_id_s                  ve_sip_call_id;
    t_ve_sip_dialog_data_s              ve_sip_dialog_data;
    t_ve_sip_subscribe_state_e          ve_sip_subscribe_state;
}t_ve_user_msg_sip_dialog_notify_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to identify the received dialog NOTIFY request.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
ve_sip_ dialog _data	Structure of type **t_ve_sip_dialog_data_s** as described in Appendix B.48 on Page 459, used to hold the dialog data.
ve_sip_subscribe_state	An enumeration type, **t_ve_sip_subscribe_state_e** as described in Appendix B.194 on Page 614, which provides the current subscription state.

B.33.35 t_ve_user_msg_sip_dialog_subscribe_response_rx_s

The **t_ve_user_msg_sip_dialog_subscribe_response_rx_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356 and provides the user with details of the received dialog SUBSCIBE response. The structure contains the session handle for the received SUBSCRIBE response as well as a SIP response code. The **t_ve_user_msg_sip_dialog_subscribe_response_rx_s** structure is defined in **ve_sip_events.h** and described below:

```
typedef struct{
    t_ve_user_handle_s                  ve_user_handle;
    t_ve_sip_session_handle_s           ve_sip_session_handle;
    t_ve_sip_call_id_s                  ve_sip_call_id;
    t_ve_sip_response_code_e            response_code;
}t_ve_user_msg_sip_dialog_subscribe_response_rx_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_sip_session_handle	Structure of type **t_ve_sip_session_handle_s**, as described in Appendix B.49 on Page 463, used to identify the received dialog SUBSCIBE response.
ve_sip_call_id	Structure of type **t_ve_sip_call_id_s**, as described in Appendix B.33.32.1 on Page 390, used to hold information on the originator of the SIP call.
response_code	An enumerated SIP response code that will be in the range of 100 to 600. The enumerated type, **t_ve_sip_response_code_e**, is defined in Appendix B.181 on Page 602.

B.33.36 t_ve_user_msg_h323_registration_response_s

The **t_ve_user_msg_h323_registration_response_s** structure is part of the structure **t_ve_user_msg_s** as defined in Appendix B.33 on Page 356 and returns an enumerated response message to a H.323 registration request. The response structure, **t_ve_user_msg_h323_registration_response_s**, is defined in **ve_h323_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s           ve_user_handle;
  t_ve_h323_reg_response_e     ve_h323_reg_response;
}t_ve_user_msg_h323_registration_response_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_h323_reg_response	An enumerated value of type **t_ve_h323_reg_response_e**, as defined in Appendix B.207 on Page 629, specifying the specific response message.

The H.323 gatekeeper will periodically update or repeat the registration response message.

B.33.37 t_ve_user_msg_h323_call_started_s

The **t_ve_user_msg_h323_call_started_s** structure is part of the structure **t_ve_user_msg_s** as defined in Appendix B.33 on Page 356 and returns the user and H.323 call handles as well as the negotiated stream configuration parameters. The response structure, **t_ve_user_msg_h323_call_started_s**, is defined in **ve_h323_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_h323_call_handle_s         ve_h323_call_handle;
  t_ve_stream_configuration_s     ve_stream_config;
  t_ve_h323_name_id_s             ve_h323_name_id;
}t_ve_user_msg_h323_call_started_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_h323_call_handle	Structure of type **t_ve_h323_call_handle_s**, as described in Appendix B.55 on Page 469, used to describe the handle of the call under consideration.
ve_stream_config	Structure of type **t_ve_stream_configuration_s**, as described in Appendix B.24 on Page 298, used to return the negotiated call parameters.
ve_h323_name_id	Structure of type **t_ve_h323_name_id_s**, as described in Appendix B.57 on Page 474, used to return the name identification of the specified call.

B.33.38 t_ve_user_msg_h323_call_stopped_s

The **t_ve_user_msg_h323_call_stopped_s** structure is part of the structure **t_ve_user_msg_s** as defined in Appendix B.33 on Page 356 and returns the user and call handles of the stopped H.323 call. The response structure, **t_ve_user_msg_h323_call_stopped_s**, is defined in **ve_h323_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s         ve_user_handle;
  t_ve_h323_call_handle_s    ve_h323_call_handle;
}t_ve_user_msg_h323_call_stopped_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_h323_call_handle	Structure of type **t_ve_h323_call_handle_s**, as described in Appendix B.55 on Page 469, used to describe the handle of the call under consideration.

B.33.39 t_ve_user_msg_h323_call_updated_s

The **t_ve_user_msg_h323_call_updated_s** structure is part of the structure **t_ve_user_msg_s** as defined in Appendix B.33 on Page 356 and returns the user and H.323 call handles as well as the updated negotiated stream configuration parameters. The response structure, **t_ve_user_msg_h323_call_updated_s**, is defined in **ve_h323_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_h323_call_handle_s         ve_h323_call_handle;
  t_ve_stream_configuration_s     ve_stream_config;
}t_ve_user_msg_h323_call_updated_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_h323_call_handle	Structure of type **t_ve_h323_call_handle_s**, as described in Appendix B.55 on Page 469, used to describe the handle of the call under consideration.
ve_stream_config	Structure of type **t_ve_stream_configuration_s**, as described in Appendix B.24 on Page 298, used to return the negotiated call parameters.

B.33.40 t_ve_user_msg_h323_call_request_s

The **t_ve_user_msg_h323_call_request_s** structure is part of the structure **t_ve_user_msg_s** as defined in Appendix B.33 on Page 356 and returns the user and call handles of the started H.323 call. The response structure, **t_ve_user_msg_h323_call_request_s**, is defined in **ve_h323_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s         ve_user_handle;
  t_ve_h323_call_handle_s    ve_h323_call_handle;
  t_ve_h323_name_id_s        ve_h323_name_id;
  Word8                      ve_call_id[VE_H323_MAX_CALL_ID_LEN];
}t_ve_user_msg_h323_call_request_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_h323_call_handle	Structure of type **t_ve_h323_call_handle_s**, as described in Appendix B.55 on Page 469, used to describe the handle of the call under consideration.
ve_h323_name_id	Structure of type **t_ve_h323_name_id_s**, as described in Appendix B.57 on Page 474, used to return the name identification of the specified call.
ve_call_id	A call identifier, such as the phone number.

B.33.41 t_ve_user_msg_h323_call_response_s

The **t_ve_user_msg_h323_call_response_s** structure is part of the structure **t_ve_user_msg_s** as defined in Appendix B.33 on Page 356 and returns an enumerated response type to a H.323 request to start a call. The response structure, **t_ve_user_msg_h323_call_response_s**, is defined in **ve_h323_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s           ve_user_handle;
  t_ve_h323_call_handle_s      ve_h323_call_handle;
  t_ve_h323_call_response_e    ve_h323_call_response;
  t_ve_h323_name_id_s          ve_h323_name_id;
  t_ve_call_waiting_flag_e     ve_call_waiting_flag;
}t_ve_user_msg_h323_call_response_s;
```

Continued >>

The elements of the **t_ve_user_msg_h323_call_response_s** structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_h323_call_handle	Structure of type **t_ve_h323_call_handle_s**, as described in Appendix B.55 on Page 469, used to describe the handle of the call under consideration.
ve_h323_call_response	An enumerated value of type **t_ve_h323_call_response_e**, as defined in Appendix B.206 on Page 626, specifying the specific response message.
ve_h323_name_id	Structure of type **t_ve_h323_name_id_s**, as described in Appendix B.57 on Page 474, used to return the name identification of the specified call.
ve_call_waiting_flag	An enumerated value of type **t-ve_h323_call_waiting_flag_e**, as defined in Appendix B.216 on Page 642, specifying if call waiting is active.

B.33.42 t_ve_user_msg_h323_transfer_response_s

The **t_ve_user_msg_h323_transfer_response_s** structure is part of the structure **t_ve_user_msg_s** as defined in Appendix B.33 on Page 356 and returns an enumerated response type to a H.323 request to transfer a call. The response structure, **t_ve_user_msg_h323_transfer_response_s**, is defined in **ve_h323_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_h323_call_handle_s         ve_h323_call_handle;
  t_ve_h323_transfer_response_e   ve_h323_transfer_response;
}t_ve_user_msg_h323_transfer_response_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_h323_call_handle	Structure of type **t_ve_h323_call_handle_s**, as described in Appendix B.55 on Page 469, used to describe the handle of the call under consideration.
ve_h323_transfer_response	An enumerated value of type **t_ve_h323_transfer_response_e**, as defined in Appendix B.210 on Page 634, specifying the specific transfer response message.

B.33.43 t_ve_user_msg_h323_call_forwarded_s

The **t_ve_user_msg_h323_call_forwarded_s** structure is part of the structure **t_ve_user_msg_s** as defined in Appendix B.33 on Page 356 and returns a response to a H.323 request to forward a call. The response structure, **t_ve_user_msg_h323_call_forwarded_s**, is defined in **ve_h323_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_h323_call_handle_s         ve_h323_call_handle;
  t_ve_h323_call_handle_s         ve_h323_call_handle_new;
}t_ve_user_msg_h323_call_forwarded_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_h323_call_handle	Structure of type **t_ve_h323_call_handle_s**, as described in Appendix B.55 on Page 469, used to describe the handle of the existing call to be forwarded.
ve_h323_call_handle_new	Structure of type **t_ve_h323_call_handle_s**, as described in Appendix B.55 on Page 469, used to describe the handle of the new call.

B.33.44 t_ve_user_msg_h323_forward_response_s

The **t_ve_user_msg_h323_forward_response_s** structure is part of the structure **t_ve_user_msg_s** as defined in Appendix B.33 on Page 356 and returns an enumerated response type to a H.323 request to activate or deactivate call forwarding. The response structure, **t_ve_user_msg_h323_forward_response_s**, is defined in **ve_h323_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s              ve_user_handle;
  t_ve_h323_call_handle_s         ve_h323_call_handle;
  t_ve_h323_forward_response_s    ve_h323_forward_response;
}t_ve_user_msg_h323_forward_response_s;
```

Continued >>

The elements of the **t_ve_user_msg_h323_call_forward_response_s** structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_h323_call_handle	Structure of type **t_ve_h323_call_handle_s**, as described in Appendix B.55 on Page 469, used to describe the handle of the existing call to be forwarded.
ve_h323_forward_response	An enumerated value of type **t_ve_h323_forward_response_e**, as defined in Appendix B.211 on Page 636, specifying the specific response message.

B.33.45 t_ve_user_msg_h323_user_input_received_s

The **t_ve_user_msg_h323_user_input_received_s** structure is part of the structure **t_ve_user_msg_s** as defined in Appendix B.33 on Page 356 and returns a character string sent from a user through the H.323 plane. The response structure, **t_ve_user_msg_h323_user_input_received_s**, is defined in **ve_h323_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s         ve_user_handle;
  t_ve_h323_call_handle_s    ve_h323_call_handle;
  Word8                      ve_digit;
}t_ve_user_msg_h323_user_input_received_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_h323_call_handle	Structure of type **t_ve_h323_call_handle_s**, as described in Appendix B.55 on Page 469, used to describe the handle of the call under consideration.
ve_digit	The single character received from the event.

B.33.46 t_ve_user_msg_h323_call_hold_response_s

The **t_ve_user_msg_h323_call_hold_response_s** structure is part of the structure **t_ve_user_msg_s** as defined in Appendix B.33 on Page 356 and returns a response code from a H.323 call hold function request. The response structure, **t_ve_user_msg_h323_call_hold_response_s**, is defined in **ve_h323_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s                ve_user_handle;
  t_ve_h323_call_handle_s           ve_h323_call_handle;
  t_ve_h323_call_hold_response_s    ve_h323_call_hold_response;
}t_ve_user_msg_h323_call_hold_response_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_h323_call_handle	Structure of type **t_ve_h323_call_handle_s**, as described in Appendix B.55 on Page 469, used to describe the handle of the call under consideration.
ve_h323_call_hold_response	An enumerated value of type **t_ve_h323_call_hold_response_e**, as defined in Appendix B.208 on Page 632, specifying the specific response message.

B.33.47 t_ve_user_msg_h323_mwi_response_s

The **t_ve_user_msg_h323_mwi_response_s** structure is part of the structure **t_ve_user_msg_s** as defined in Appendix B.33 on Page 356 and returns a response code from a H.323 message waiting function request. The response structure, **t_ve_user_msg_h323_mwi_response_s**, is defined in **ve_h323_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s           ve_user_handle;
  t_ve_h323_mwi_response_s     ve_h323_mwi_response;
}t_ve_user_msg_h323_mwi_response_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_h323_mwi_response	An enumerated value of type **t_ve_h323_mwi_response_e**, as defined in Appendix B.209 on Page 633, specifying the specific response message.

B.33.48 t_ve_user_msg_h323_message_waiting_s

The **t_ve_user_msg_h323_message_waiting_s** structure is part of the structure **t_ve_user_msg_s** as defined in Appendix B.33 on Page 356 and returns details of calls waiting for the specified H.323 user. The response structure, **t_ve_user_msg_h323_message_waiting_s**, is defined in **ve_h323_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s         ve_user_handle;
  UWord16                    num_messages;
  Word8                      service[VE_H323_MAX_SERVICE_LEN];
  t_ve_h323_msg_center_id_s  msg_center_id;
  t_ve_h323_dest_address_s   origin_address;
  Word8                      time[VE_H323_MAX_TIME_LEN];
  Word32                     priority;
}t_ve_user_msg_h323_message_waiting_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
num_messages	The number of messages waiting at the message center.
service	The type of service for which a message is waiting.
msg_center_id	A structure of type **t_ve_h323_msg_center_id_s**, as described in Appendix B.53 on Page 468, that provides the unique ID of the Message Center that initiated the service.
origin address	Structure of type **t_ve_h323_dest_address_s**, as described in Appendix B.54 on Page 469, used to provide the message originators IP address.
time	A timestamp associated to the message.
priority	The priority of the waiting message.

The exact format of Message Center specific elements of this structure are dependent on the Message Center application. Please refer to the Message Center documentation for this information.

B.33.49 t_ve_user_msg_h323_request_failed_s

The **t_ve_user_msg_h323_request_failed_s** structure is part of the structure **t_ve_user_msg_s** as defined in Appendix B.33 on Page 356 and returns an enumerated reason type why a legal VeriCall Edge H.323 API call failed to be processed. The response structure, **t_ve_user_msg_h323_request_failed_s**, is defined in **ve_h323_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s                ve_user_handle;
  t_ve_h323_call_handle_s           ve_h323_call_handle;
  t_ve_h323_request_type_e          request_type;
  t_ve_h323_request_failed_reason_e reason;
}t_ve_user_msg_h323_request_failed_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_h323_call_handle	Structure of type **t_ve_h323_call_handle_s**, as described in Appendix B.55 on Page 469, used to describe the handle of the call under consideration.
request_type	An enumerated value of type **t_ve_h323_request_type_e**, as defined in Appendix B.212 on Page 637, specifying the type of request that failed. For example, if *request_type* is VE_H323_REQUEST_TYPE_VE_H323STARTCALL then there was a call to **VE_H323StartCall** that failed.
reason	An enumerated value of type **t_ve_h323_request_failed_reason_e**, as defined in Appendix B.213 on Page 639, specifying the reason for the failure.

B.33.50 t_ve_user_msg_h323_video_feature_request_s

The **t_ve_user_msg_h323_video_feature_request_s** structure is part of the structure **t_ve_user_msg_s** as defined in Appendix B.33 on Page 356 and returns a response code from a H.323 video command function. The notification event structure, **t_ve_user_msg_h323_video_feature_request_s**, is defined in **ve_h323_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s                ve_user_handle;
  t_ve_h323_call_handle_s           ve_h323_call_handle;
  t_ve_h323_video_feature_request_e ve_h323_video_feature_request;
}t_ve_user_msg_h323_video_feature_request_s;
```

The elements of this structure are as follows:

Element	Definition
ve_user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
ve_h323_call_handle	Structure of type **t_ve_h323_call_handle_s**, as described in Appendix B.55 on Page 469, used to describe the handle of the call under consideration.
ve_h323_video_feature_request	An enumerated value of type **t_ve_h323_video_feature_request_e**, as defined in Appendix B.219 on Page 645, specifying the specific response message.

B.33.51 t_ve_user_msg_detect_tone_dtmf_s

The **t_ve_user_msg_detect_tone_dtmf_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The **t_ve_user_msg_detect_tone_dtmf_s** structure provides per-channel detected DTMF tone information to the user application. The **t_ve_user_msg_detect_tone_dtmf_s** structure is defined in **ve_events.h** and is described below:

```
typedef struct{
  Word8                          tone_value;
  UWord32                        channel;
  t_ve_user_sub_channel_select_e sub_channel_select;
}t_ve_user_msg_detect_tone_dtmf_s;
```

The elements of this structure are as follows:

Element	Definition
tone_value	The DTMF tone detected, as a character.
channel	The channel number that the DTMF tone was detected on.
sub_channel_select	Reserved for future use.

B.33.52 t_ve_user_msg_detect_tone_cp_s

The **t_ve_user_msg_detect_tone_cp_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The **t_ve_user_msg_detect_tone_cp_s** structure provides per-channel detected Call Progress tone information to the user application. The **t_ve_user_msg_detect_tone_cp_s** structure is defined in **ve_events.h** and is described below:

```
typedef struct{
  t_ve_country_e                  tone_country_type;
  t_ve_tone_type_e                tone_type;
  UWord32                         channel;
  t_ve_user_sub_channel_select_e  sub_channel_select;
}t_ve_user_msg_detect_tone_cp_s;
```

The elements of this structure are as follows:

Element	Definition
tone_country_type	The enumerated country type of the detected Call Progress tone, as described in Appendix B.79 on Page 518.
tone_type	The enumerated Call Progress tone type, as described in Appendix B.78 on Page 517.
channel	The channel number that the Call Progress tone was detected on.
sub_channel_select	An enumerated type, **t_ve_user_sub_channel_select_e** as described in Appendix B.167 on Page 593, used to determine the sub-channel on which the detection occurred.

B.33.53 t_ve_user_msg_detect_tone_2100_s

The **t_ve_user_msg_detect_tone_2100_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The **t_ve_user_msg_detect_tone_2100_s** structure provides per-channel detected 2100Hz tone information to the user application. The **t_ve_user_msg_detect_tone_2100_s** structure is defined in **ve_events.h** and is described below:

```
typedef struct{
  UWord32                         channel;
  t_ve_user_sub_channel_select_e  sub_channel_select;
}t_ve_user_msg_detect_tone_2100_s;
```

The element of this structure is as follows:

Element	Definition
channel	The channel number that the 2100Hz tone was detected on.
sub_channel_select	An enumerated type, **t_ve_user_sub_channel_select_e** as described in Appendix B.167 on Page 593, used to determine the sub-channel on which the detection occurred.

B.33.54 t_ve_user_msg_detect_tone_1100_s

The **t_ve_user_msg_detect_tone_1100_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The **t_ve_user_msg_detect_tone_1100_s** structure provides per-channel detected 1100Hz tone information to the user application. The **t_ve_user_msg_detect_tone_1100_s** structure is defined in **ve_events.h** and is described below:

```
typedef struct{
  UWord32                          channel;
  t_ve_user_sub_channel_select_e   sub_channel_select;
}t_ve_user_msg_detect_tone_1100_s;
```

The element of this structure is as follows:

Element	Definition
channel	The channel number that the 1100Hz tone was detected on.
sub_channel_select	An enumerated type, **t_ve_user_sub_channel_select_e** as described in Appendix B.167 on Page 593, used to determine the sub-channel on which the detection occurred.

B.33.55 t_ve_user_msg_received_rfc2833_digit_s

The **t_ve_user_msg_received_rfc2833_digit_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The **t_ve_user_msg_received_rfc2833_digit_s** structure provides per-channel detected RFC2833 digit information to the user application. The **t_ve_user_msg_received_rfc2833_digit_s** structure is defined in **ve_events.h** and is described below:

```
typedef struct{
  UWord8                           tone_value;
  UWord32                          channel;
  t_ve_user_sub_channel_select_e   sub_channel_select;
}t_ve_user_msg_detect_tone_2100_s;
```

The element of this structure is as follows:

Element	Definition
tone_value	The RFC2833 digit detected, as a character.
channel	The channel number that the RFC2833 digit was detected on.
sub_channel_select	An enumerated type, **t_ve_user_sub_channel_select_e** as described in Appendix B.167 on Page 593, used to determine the sub-channel on which the detection occurred.

B.33.56 t_ve_user_msg_play_prompt_event_s

The **t_ve_user_msg_play_prompt_event_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The **t_ve_user_msg_play_prompt_event_s** structure provides per-channel event notification for a play into audio stream command issued from the user application. The **t_ve_user_msg_play_prompt_event_s** structure is defined in **ve_events.h** and is described below:

```
typedef struct{
  t_ve_user_msg_play_prompt_event_e    event;
  UWord32                              channel;
  t_ve_user_sub_channel_select_e       sub_channel_select;
}t_ve_user_msg_play_prompt_event_s;
```

The elements of this structure are as follows:

Element	Definition
event	The enumerated event type. The enumeration, **t_ve_user_msg_play_prompt_event_e**, is defined in Appendix B.223 on Page 648.
channel	The channel number that the event was raised on.
sub_channel_select	An enumerated type, **t_ve_user_sub_channel_select_e** as described in Appendix B.167 on Page 593, used to determine the sub-channel on which the event occurred.

B.33.57 t_ve_user_msg_record_prompt_event_s

The **t_ve_user_msg_record_prompt_event_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The **t_ve_user_msg_record_prompt_event_s** structure provides per-channel event notification for a record audio stream command issued from the user application. The **t_ve_user_msg_record_prompt_event_s** structure is defined in **ve_events.h** and is described below:

```
typedef struct{
  t_ve_user_msg_record_prompt_event_e  event;
  UWord32                              channel;
  t_ve_user_sub_channel_select_e       sub_channel_select;
}t_ve_user_msg_record_prompt_event_s;
```

The elements of this structure are as follows:

Element	Definition
event	The enumerated event type. The enumeration, **t_ve_user_msg_record_prompt_event_e,** is defined in Appendix B.224 on Page 649.
channel	The channel number that the event was raised on.
sub_channel_select	An enumerated type, **t_ve_user_sub_channel_select_e** as described in Appendix B.167 on Page 593, used to determine the sub-channel on which the event occurred.

B.33.58 t_ve_user_msg_encode_image_event_s

The **t_ve_user_msg_encode_image_event_s** structure is part of the **t_ve_user_msg_s** structure, as described in Appendix B.33 on Page 356, and returns a response code from a H.323 video encode image command function. The notification event structure, **t_ve_user_msg_encode_image_event_s**, is defined in **ve_events.h** and is described below:

```
typedef struct{
  t_ve_encode_image_event_type_e        image_event_type;
  t_ve_user_msg_encode_image_event_e    event_reason;
  UWord32                               channel;
}t_ve_user_msg_encode_image_event_s;
```

The element of this structure is as follows:

Element	Definition
image_event_type	Enumerated type, **t_ve_encode_image_event_type_e** as described in Appendix B.226 on Page 650, which specifies whether the completion event refers to a local or remote image event.
event_reason	Enumerated type, **t_ve_user_msg_encode_image_event_e** as described in Appendix B.225 on Page 649, which provides the response code for this event notification.
channel	The channel number against which this event was raised.

B.33.59 t_ve_user_msg_video_encode_path_config_err_s

The **t_ve_user_msg_video_encode_path_config_err_s** structure is part of the **t_ve_user_msg_s** structure, as described in Appendix B.33 on Page 356. This event is generated when the video encoder is incorrectly configured. The structure contains the video encoder configuration that caused the error event to be generated. The notification event structure, **t_ve_user_msg_video_encode_path_config_err_s**, is defined in **ve_events.h** and is described below:

```
typedef struct{
  t_ve_video_config_error_reason_e       reason;
  UWord32                  channel;
  UWord32                  pre_local_image_h_origin;
  UWord32                  pre_local_image_v_origin;
  UWord32                  pre_local_image_h_size;
  UWord32                  pre_local_image_v_size;
  UWord32                  pre_local_image_h_scale_factor_num;
  UWord32                  pre_local_image_h_scale_factor_den;
  UWord32                  pre_local_image_v_scale_factor_num;
  UWord32                  pre_local_image_v_scale_factor_den;
  t_ve_video_default_image_rotate_e      pre_local_image_rotation;
  t_ve_video_default_image_mirror_e      pre_local_image_mirror;
  UWord32                  pre_local_image_data_end_point_instance;
  UWord32                  pre_camera_h_size;
  UWord32                  pre_camera_v_size;
  UWord32                  pre_camera_data_end_point_instance;
  UWord32                  pre_tx_image_h_size;
  UWord32                  pre_tx_image_v_size;
  UWord32                  pre_tx_image_h_scale_factor_num;
  UWord32                  pre_tx_image_h_scale_factor_den;
  UWord32                  pre_tx_image_v_scale_factor_num;
  UWord32                  pre_tx_image_v_scale_factor_den;
  t_ve_video_default_image_rotate_e      pre_tx_image_rotation;
  t_ve_video_default_image_mirror_e      pre_tx_image_mirror;
  UWord32                  enc_image_h_size;
  UWord32                  enc_image_v_size;
}t_ve_user_msg_video_encode_path_config_err_s;
```

The element of this structure is as follows:

Element	Definition
reason	Enumerated type, **t_ve_video_config_error_reason_e** as described in Appendix B.157 on Page 580, which specifies the reason for this event being generated.
channel	The channel number against which this event was raised.
pre_local_image_h_origin	The local image horizontal origin.
pre_local_image_v_origin	The local image vertical origin.
pre_local_image_h_size	The local image horizontal size.
pre_local_image_v_size	The local image vertical size.
pre_local_image_h_scale_factor_num	The local image horizontal scale factor numerator.

pre_local_image_h_scale_factor_den	The local image horizontal scale factor denominator.
pre_local_image_v_scale_factor_num	The local image vertical scale factor numerator.
pre_local_image_v_scale_factor_den	The local image vertical scale factor denominator.
pre_local_image_rotation	Enumerated type, **t_ve_video_default_image_rotate_e** as described in Appendix B.95 on Page 530, which specifies the local image rotation setting before pre-processing.
pre_local_image_mirror	Enumerated type, **t_ve_video_default_image_mirror_e** as described in Appendix B.96 on Page 531, which specifies the local image mirror setting before pre-processing.
pre_local_image_data_end_point_instance	The local image data end-point instance.
pre_camera_h_size	The local camera horizontal size.
pre_camera_v_size	The local camera vertical size.
pre_camera_data_end_point_instance	The local camera data end-point instance.
pre_tx_image_h_size	The horizontal size numerator of the image to be transmitted.
pre_tx_image_v_size	The vertical size numerator of the image to be transmitted.
pre_tx_image_h_scale_factor_num	The horizontal scale factor numerator of the image to be transmitted.
pre_tx_image_h_scale_factor_den	The horizontal scale factor denominator numerator of the image to be transmitted.
pre_tx_image_v_scale_factor_num	The vertical scale factor numerator of the image to be transmitted.
pre_tx_image_v_scale_factor_den	The vertical scale factor denominator numerator of the image to be transmitted.
pre_tx_image_rotation	Enumerated type, **t_ve_video_default_image_rotate_e** as described in Appendix B.95 on Page 530, which specifies the rotation setting of the image to be transmitted.
pre_tx_image_mirror	Enumerated type, **t_ve_video_default_image_mirror_e** as described in Appendix B.96 on Page 531, which specifies the image mirror setting of the image to be transmitted
enc_image_h_size	The horizontal size of the image after encoding.
enc_image_v_size	The vertical size of the image after encoding.

B.33.60 t_ve_user_msg_stream_event_s

The **t_ve_user_msg_stream_event_s** structure is part of the **t_ve_user_msg_s** structure, as described in Appendix B.33 on Page 356, and returns a response code from a stream command function. The notification event structure, **t_ve_user_msg_stream_event_s**, is defined in **ve_events.h** and is described below:

```
typedef struct{
  t_ve_user_handle_s                    handle;
  UWord32                               channel;
  t_ve_user_sub_channel_select_e        sub_channel_select;
  t_ve_user_msg_stream_event_reason_e   reason;
  t_ve_user_msg_stream_event_type_e     type;
}t_ve_user_msg_stream_event_s;
```

The element of this structure is as follows:

Element	Definition
handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
channel	The channel number against which this event was raised.
sub_channel_select	Enumerated type, **t_ve_user_sub_channel_index_e** as described in Appendix B.168 on Page 594, which specifies the sub-channel on which the event was detected on.
reason	Enumerated type, **t_ve_user_msg_stream_event_reason_e** as described in Appendix B.163 on Page 589, which specifies the reason for this event being generated.
type	Enumerated type, **t_ve_user_msg_stream_event_type_e** as described in Appendix B.164 on Page 590, which specifies whether this event was due to a start/update stream (**VE_StartStream** and **VE_UpdateStream**) or a stop stream (**VE_StopStream**).

B.33.61 t_ve_user_msg_module_state_transition_event_s

The **t_ve_user_msg_module_state_transition_event_s** structure is part of the **t_ve_user_msg_s** structure, as described in Appendix B.33 on Page 356, and returns a response code from a class modules state transition. This response will only be returned due to an error, normally when there is a audio/video call active and a request has been made to reduce resource usage, The request will be rejected and this error response returned. The notification event structure, **t_ve_user_msg_module_state_transition_event_s**, is defined in **ve_events.h** and is described below:

```
typedef struct{
  t_ve_user_msg_module_state_transition_reason_e    reason;
}t_ve_user_msg_module_state_transition_event_s;
```

The element of this structure is as follows:

Element	Definition
reason	Enumerated type, **t_ve_user_msg_module_state_transition_reason_e** as described in Appendix B.165 on Page 590, which specifies the reason for this event being generated.

B.33.62 t_ve_user_msg_alarm_s

The **t_ve_user_msg_alarm_s** structure is part of the **t_ve_user_msg_s** structure defined in Appendix B.33 on Page 356. The **t_ve_user_msg_alarm_s** structure provides VeriCall Edge generated alarms. This structure is defined in **ve_events.h** and the associated unions are defined in **ve_alarms.h** and are described below:

```
typedef struct{
  t_ve_alarm_type_e          alarm_type;
  t_ve_alarm_subtype_u       alarm_subtype;
  t_ve_alarms_u              alarm;
}t_ve_user_msg_alarm_s;

typedef union{
  t_ve_alarm_system_subtype_e     system_subtype;
  t_ve_alarm_sip_subtype_e        sip_subtype;
  t_ve_alarm_h323_subtype_e       h323_subtype;
}t_ve_alarm_subtype_u;

typedef union{
  t_ve_alarm_subtype_ajbm_discard_buffer_full_s
                                     ve_alarm_sys_ajbm_discard;
  t_ve_alarm_subtype_ajbm_extended_break_in_stream_s
                                     ve_alarm_sys_ajbm_stream_break;
  t_ve_alarm_subtype_ajbm_underrun_s
                                     ve_alarm_sys_ajbm_underrun;
  t_ve_alarm_subtype_sip_alarm_s     ve_alarm_sip;
  t_ve_alarm_subtype_h323_alarm_s    ve_alarm_h323;
}t_ve_alarms_u;
```

The elements of this structure are as follows:

Element	Definition
alarm_type	The enumerated type of alarm as described in Appendix B.158 on Page 581.
alarm_subtype	A union of alarm subtypes. The subtypes can be one of the following: **t_ve_alarm_system_subtype_e** enumerated type as described in Appendix B.159 on Page 581. **t_ve_alarm_sip_subtype_e** enumerated type as described in Appendix B.160 on Page 582. **t_ve_alarm_h323_subtype_e** enumerated type as described in Appendix B.161 on Page 583.
alarm	Union of alarm structures that can be one of the following: **t_ve_alarm_subtype_ajbm_discard_buffer_full_s** as described in Appendix B.33.62.1 on Page 414. **t_ve_alarm_subtype_ajbm_extended_break_in_stream_s** as described in Appendix B.33.62.2 on Page 414. **t_ve_alarm_subtype_ajbm_underrun_s** as described in Appendix B.33.62.3 on Page 415. **t_ve_alarm_subtype_sip_alarm_s** as described in Appendix B.33.62.4 on Page 416. **t_ve_alarm_subtype_h323_alarm_s** as described in Appendix B.33.62.5 on Page 416.

B.33.62.1 t_ve_alarm_subtype_ajbm_discard_buffer_full_s

The **t_ve_alarm_subtype_ajbm_discard_buffer_full_s** structure is part of the **t_ve_user_msg_alarm_s** structure as described in Appendix B.33.58 on Page 408. The **t_ve_alarm_subtype_ajbm_discard_buffer_full_s** structure provides the alarm to indicate that the AJBM buffer has discarded packets due to the buffer being full. The structure is defined in **ve_alarms.h** and is described below:

```
typedef struct{
  UWord32    reserved;
}t_ve_alarm_subtype_ajbm_discard_buffer_full_s;
```

The element of this structure is as follows:

Element	Definition
reserved	Reserved for future use.

For information on suggested settings for the VeriCall Edge AJBM, refer to the t_ve_ajbm_algorithm_default_configuration_s structure described in Appendix B.9 on Page 262.

B.33.62.2 t_ve_alarm_subtype_ajbm_extended_break_in_stream_s

The **t_ve_alarm_subtype_ajbm_extended_break_in_stream_s** structure is part of the **t_ve_user_msg_alarm_s** structure as described in Appendix B.33.58 on Page 408. The **t_ve_alarm_subtype_ajbm_extended_break_in_stream_s** structure provides the alarm to indicate that the no data or SID packet has been received by the AJBM within the expected time. The structure is defined in **ve_alarms.h** and is described below:

```
typedef struct{
  UWord32    reserved;
}t_ve_alarm_subtype_ajbm_extended_break_in_stream_s;
```

The element of this structure is as follows:

Element	Definition
reserved	Reserved for future use.

B.33.62.3 t_ve_alarm_subtype_ajbm_underrun_s

The **t_ve_alarm_subtype_ajbm_underrun_s** structure is part of the **t_ve_user_msg_alarm_s** structure as described in Appendix B.33.58 on Page 408. The **t_ve_alarm_subtype_ajbm_underrun_s** structure provides the alarm to indicate that the AJBM has not received sufficient packets. The structure is defined in **ve_alarms.h** and is described below:

```
typedef struct{
  UWord32    reserved;
}t_ve_alarm_subtype_ajbm_underrun_s;
```

The element of this structure is as follows:

Element	Definition
reserved	Reserved for future use.

B.33.62.4 t_ve_alarm_subtype_sip_alarm_s

The **t_ve_alarm_subtype_sip_alarm_s** structure is part of the **t_ve_user_msg_alarm_s** structure as described in Appendix B.33.58 on Page 408. The **t_ve_alarm_subtype_sip_alarm_s** structure provides the alarm to indicate a SIP alarm occurrence. The structure is defined in **ve_sip_alarms.h** and is described below:

```
typedef struct{
  UWord32    reserved;
}t_ve_alarm_subtype_sip_alarm_s;
```

The elements of this structure are as follows:

Element	Definition
reserved	Reserved for future use.

B.33.62.5 t_ve_alarm_subtype_h323_alarm_s

The **t_ve_alarm_subtype_h323_alarm_s** structure is part of the **t_ve_user_msg_alarm_s** structure as described in Appendix B.33.58 on Page 408. The **t_ve_alarm_subtype_h323_alarm_s** structure provides the alarm to indicate a H.323 alarm occurrence. The structure is defined in **ve_h323_alarms.h** and is described below:

```
typedef struct{
  UWord32    reserved;
}t_ve_alarm_subtype_h323_alarm_s;
```

The elements of this structure are as follows:

Element	Definition
reserved	Reserved for future use.

B.33.63 t_ve_user_msg_shutdown_complete_s

The **t_ve_user_msg_shutdown_complete_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The **t_ve_user_msg_shutdown_complete_s** structure provides an indication of when the VeriCall Edge system has successfully shutdown, after a **VE_Shutdown** call. Only after receiving this message should the user application exit. The **t_ve_user_msg_shutdown_complete_s** structure is defined in **ve_events.h** and is described below:

```
typedef struct{
  Word32        reserved;
}t_ve_user_msg_shutdown_complete_s;
```

The element of this structure is as follows:

Element	Definition
reserved	This element is reserved for future use.

B.33.64 t_ve_user_msg_network_connection_event_s

The **t_ve_user_msg_network_connection_event_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The **t_ve_user_msg_network_connection_event_s** structure provides information on a network connection for use by the NAT traversal module. This event is raised when VeriCall Edge creates a network connection to a remote peer. The **t_ve_user_msg_network_connection_event_s** structure is defined in **ve_events.h** and is described below:

```
typedef struct{
  t_ve_network_connection_event_e      event;
  t_ve_network_connection_s            connection;
}t_ve_user_msg_network_connection_event_s;
```

The element of this structure is as follows:

Element	Definition
event	Enumerated type, **t_ve_network_connection_event_e** as described in Appendix B.136 on Page 562, which specifies the network connection event that generated this message event.
connection	Structure of type **t_ve_network_connection_s**, as described in Appendix B.64 on Page 498, used to indicate the network connection that generated this event.

B.33.65 t_ve_user_msg_net_lookup_response_s

The **t_ve_user_msg_net_lookup_response_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. This structure provides the result from a **VE_NETLookUp** call. The **t_ve_user_msg_net_lookup_response_s** structure is defined in **ve_net_events.h** and is described below:

```
typedef struct{
  t_ve_net_lookup_query_s      query;
  UWord32                      num_records;
  t_ve_net_lookup_record_s     records[VE_NET_MAX_LOOKUP_RECORDS];
}t_ve_user_msg_net_lookup_response_s;
```

The element of this structure is as follows:

Element	Definition
query	Structure of type **t_ve_net_lookup_query_s**, as described in Appendix B.66 on Page 500, that contains the original network lookup query.
num_records	The number of result records.
records	Array of structures, of type **t_ve_net_lookup_record_s** as described in Appendix B.33.65.1 on Page 419, used to contain the list of result records in preference order. The maximum number of records returnable cannot exceed `VE_NET_MAX_LOOKUP_RECORDS`.

B.33.65.1 t_ve_net_lookup_record_s

The **t_ve_net_lookup_record_s** structure is used within the **t_ve_user_msg_net_lookup_response_s** structure, described in Appendix B.33.65 on Page 418. This structure contains the network lookup result information from a **VE_NETLookUp** call. This structure is defined in **ve_net_defs.h** and is described below:

```
typedef struct{
  t_ve_net_lookup_record_type_e                type;
  union{
     t_ve_net_lookup_record_ipv4_address_s     ipv4_address;
     t_ve_net_lookup_record_server_s           server;
     t_ve_net_lookup_record_naptr_s            naptr;
  }data;
}t_ve_net_lookup_record_s;
```

The elements of this structure are as follows:

Element	Definition
type	An enumerated value, of type **t_ve_net_lookup_record_type_e** as described in Appendix B.140 on Page 566, used to determine the network lookup type.
data	A union of structures whose selection depends on the value of the *type* element. The selected structure can be one of the following: • Structure **t_ve_net_lookup_ipv4_address_s** as Described in Appendix B.67 on Page 501. This structure defines the IPv4 address looked up result if requested. • Structure **t_ve_net_lookup_server_s** as described in Appendix B.68 on Page 501. This structure defines the server type and domain result if requested. • Structure **t_ve_net_lookup_naptr_s** as described in Appendix B.69 on Page 502. This structure defines the NAPTR lookup result if requested.

B.33.65.2 t_ve_net_lookup_record_ipv4_address_s

The **t_ve_net_lookup_record_ipv4_address_s** structure is used within the **t_ve_net_lookup_record_s** structure, described in Appendix B.33.65.1 on Page 419. This structure contains the network address information from a **VE_NETLookUp** call. This structure is defined in **ve_net_defs.h** and is described below:

```
typedef struct{
  UWord32       time_to_live;
  UWord32       address;
}t_ve_net_lookup_record_ipv4_address_s;
```

The elements of this structure are as follows:

Element	Definition
time_to_live	The time to live value as per RFC1035.
address	The host IPv4 address in network byte order.

The VeriCall Edge system uses the network byte ordering for IP addresses when a UWord32 value is required.

B.33.65.3 t_ve_net_lookup_record_server_s

The **t_ve_net_lookup_record_server_s** structure is used within the **t_ve_net_lookup_record_s** structure, described in Appendix B.33.65.1 on Page 419. This structure contains information about the network lookup server information requested through a **VE_NETLookUp** call. This structure is defined in **ve_net_defs.h** and is described below:

```
typedef struct{
  UWord32       time_to_live;
  UWord16       priority;
  UWord16       weight;
  UWord16       port;
  UWord8        target[VE_MAX_URI_STRING_SIZE];
}t_ve_net_lookup_record_server_s;
```

The elements of this structure are as follows:

Element	Definition
time_to_live	The time to live as per RFC 1035.
priority	The priority setting as per RFC2782.
weight	The weight value as per RFC2782.
port	The port number as per RFC2782.
target	The fully qualified resource name.

B.33.65.4 t_ve_net_lookup_naptr_s

The **t_ve_net_lookup_naptr_s** structure is used within the **t_ve_net_lookup_record_s** structure, described in Appendix B.33.65.1 on Page 419. This structure contains the NAPTR lookup result from a **VE_NETLookUp** call. This structure is defined in **ve_net_defs.h** and is described below:

```
typedef struct{
  UWord32       time_to_live;
  UWord16       order;
  UWord16       preference;
  UWord8        resource[VE_MAX_URI_STRING_SIZE];
}t_ve_net_lookup_record_naptr_s;
```

The elements of this structure are as follows:

Element	Definition
time_to_live	The time to live as per RFC1035.
priority	The order value as defined per RFC2915.
weight	The preference value as per RFC2915.
resource	The fully qualified resource name.

B.33.66 t_ve_user_msg_net_request_failed_s

The **t_ve_user_msg_net_request_failed_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. This structure provides details of a network request failure that has arisen from a **VE_NETLookUp** call. The **t_ve_user_msg_net_request_failed_s** structure is defined in **ve_net_events.h** and is described below:

```
typedef struct{
  t_ve_net_request_type_e               request_type;
  t_ve_net_request_failed_reason_e      reason;
  UWord32                               transaction_id;
}t_ve_user_msg_net_request_failed_s;
```

The element of this structure is as follows:

Element	Definition
request_type	Enumerated type, **t_ve_net_request_type_e** as described in Appendix B.141 on Page 566, which specifies the type of network request that failed.
reason	Enumerated type, **t_ve_net_request_failed_reason_e** as described in Appendix B.142 on Page 567, which specifies the reason for the network lookup failure.
transaction_id	The client transaction identifier.

B.33.67 t_ve_user_msg_undecodeable_stream_s

The **t_ve_user_msg_undecodeable_stream_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. This structure provides details of a received RTP stream which is invalid or does not match what the decoder is expecting. The **t_ve_user_msg_undecodeable_stream_s** structure is defined in **ve_events.h** and is described below:

```
typedef struct {
  t_ve_user_handle_s                              user_handle;
  UWord32                                         channel;
  t_ve_user_sub_channel_select_e                  sub_channel_select;
  t_ve_media_id_e                                 media_id;
  t_ve_user_msg_undecodeable_stream_reason_e      reason;
}t_ve_user_msg_undecodeable_stream_s;
```

The elements of this structure is as follows:

Element	Definition
user_handle	Structure of type **t_ve_user_handle_s**, as described in Appendix B.25 on Page 299, used to describe the handle of the user under consideration.
channel	The channel number that the event was raised on.
sub_channel_select	Enumerated type, **t_ve_user_sub_channel_select_e** as described in Appendix B.167 on Page 593, which specifies the sub-channel type the response is associated with.
media_id	The ID of the media profile type, as defined in the **t_ve_media_id_e** enumeration described in Appendix B.110 on Page 539, within the channel that encountered the error.
reason	Enumerated type, **t_ve_user_msg_undecodeable_stream_reason_e** as described in Appendix B.233 on Page 656, which specifies the reason for the error.

B.33.68 t_ve_user_msg_decode_media_change_event_s

The **t_ve_user_msg_decode_media_change_event_s** structure is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. This structure provides details of a change to the decoding stream of a channel. The **t_ve_user_msg_decode_media_change_event_s** structure is defined in **ve_events.h** and described below:

```
typedef struct{
  t_ve_media_id_e                 media_id;
  UWord32                         channel;
  t_ve_user_sub_channel_select_e  sub_channel_select;
}t_ve_user_msg_decode_media_change_event_s;
```

The elements of this structure is as follows:

Element	Definition
media_id	Enumerated type **t_ve_media_id_e,** as described in Appendix B.110 on Page 539, that identifies the new media being received on the decode path.
channel	The channel number the event is associated with.
sub_channel_select	**t_ve_user_sub_channel_select_e,** as described in Appendix B.167 on Page 593, that identifies the sub-channel of the changed inbound stream.

B.34 t_ve_callback_data_channel_descriptor_s

The **t_ve_callback_data_channel_descriptor_s** structure is used by the **VE_RegisterChannelDataSourceSinkOpen** function. The structure defines the type of descriptor to be used per-channel, specifically the data-side of the channel. This structure is defined in **ve_callback.h** and is described below:

```
typedef struct{
  Word8   descriptor[VE_CALLBACK_MAX_DESCRIPTOR_STR_LEN];
}t_ve_callback_data_channel_descriptor_s;
```

The elements of this structure is as follows:

Element	Definition
descriptor	File descriptor of the device terminating this callback function. The file descriptor must not exceed `VE_CALLBACK_MAX_DESCRIPTOR_STR_LEN` in length (defined as being 100 characters including termination character).

B.35 t_ve_callback_host_descriptor_s

The **t_ve_callback_host_descriptor_s** structure is used by the **VE_RegisterPacketTransportOpen** function to define the transport type and physical address for the transmission/reception of packet data. This structure is defined in **ve_callback.h** and is described below:

```
typedef struct{
  t_ve_callback_transport_type_e    type;
  union{
     UWord32    ipv4;
     Word8      fqdn[VE_CALLBACK_MAX_FQDN_STR_LEN];
  }addr;
}t_ve_callback_host_descriptor_s;
```

The enumeration elements are as follows:

Element	Definition
type	Indicates the transport type used and is an enumerated type of **t_ve_callback_transport_type_e** as described in Appendix B.170 on Page 595.
addr	A union of elements that dependent on type can be one of the following: **ipv4** An IPv4 address. **fqdn** A NULL terminated fully qualified domain name that must not exceed VE_CALLBACK_MAX_FQDN_STR_LEN (currently defined as being 100 characters including the termination character).

The VeriCall Edge system uses the network byte ordering for IP addresses when a UWord32 value is required.

B.36 t_ve_callback_device_query_u

The **t_ve_callback_device_query_u** union is used by callback functions registered through **VE_RegisterChannelDataDeviceQuery**. The **t_ve_callback_device_query_u** is defined in **ve_callback.h** and is described below:

```
typedef union{
  t_ve_callback_audio_device_query_s           audio_dev;
  t_ve_callback_video_display_device_query_s   video_display_dev;
  t_ve_callback_video_capture_device_query_s   video_capture_dev;
}t_ve_callback_device_query_u;
```

The structures within this union are as follows:

Element	Definition
audio_dev	A structure, **t_ve_callback_audio_device_query_s** as defined in Appendix B.36.1 on Page 427, used to determine the configuration properties of an audio device.
video_display_dev	A structure, **t_ve_callback_video_display_device_query_s** as defined in Appendix B.36.2 on Page 428, used to determine the configuration properties of a video display device.
video_capture_dev	A structure, **t_ve_callback_video_capture_device_query_s** as defined in Appendix B.36.3 on Page 429, used to determine the configuration properties of a video capture device.

B.36.1 t_ve_callback_audio_device_query_s

The **t_ve_callback_audio_device_query_s** structure is used within the **t_ve_callback_device_query_u** union as defined in Appendix B.36 on Page 426. The **t_ve_callback_audio_device_query_s** structure allows for the storage of specific information from a queried audio device. The structure is defined in **ve_callback.h** and is described below:

```
typedef struct{
  UWord32                           sample_rate;
  UWord32                           buffer_depth;
  t_ve_callback_duplex_capability_e duplex;
  UWord32                           max_latency;
}t_ve_callback_audio_device_query_s;
```

The elements of the **t_ve_callback_audio_device_query_s** structure are as follows:

Element	Definition
sample_rate	The sample rate of this audio device.
buffer_depth	The buffer depth of this audio device that can be written to or read from.
duplex	An enumerated type, **t_ve_callback_duplex_capability_e** as defined in Appendix B.176 on Page 598, that determines whether this audio device has half or full duplex capabilities.
max_latency	The maximum latency of the data being received.

B.36.2 t_ve_callback_video_display_device_query_s

The **t_ve_callback_video_display_device_query_s** structure is used within the **t_ve_callback_device_query_u** union as defined in Appendix B.36 on Page 426. The **t_ve_callback_video_display_device_query_s** structure allows for the storage of specific information from a queried video display device. The structure is defined in **ve_callback.h** and is described below:

```
typedef struct{
  UWord32                                frame_rate;
  UWord32                                frame_buffer_address;
  UWord32                                image_width;
  UWord32                                image_height;
  UWord32                                image_stride;
  t_ve_callback_video_color_format_e     color_format;
  t_ve_callback_video_frame_buffer_e     frame_buffer_type;
  UWord16                                transparency;
}t_ve_callback_video_display_device_query_s;
```

The elements of this structure are as follows:

Element	Definition
frame_rate	The frame rate capability of the video display device.
frame_buffer_address	The address of the video display device buffer to be written to.
image_width	The horizontal size of the video image display.
image_height	The vertical size of the video image display.
image_stride	The stride to be used when writing to this video image display.
color_format	An enumerated type, **t_ve_callback_video_color_format_e** as defined in Appendix B.175 on Page 598. The enumerated type determines the color format supported by the video image display device.
frame_buffer_type	An enumerated type, **t_ve_callback_video_frame_buffer_e** as described in Appendix B.178 on Page 599, used to specify the Data Manager interface buffer type to be used.
transparency	Allows a RGB-16 color to be defined as being the selected secondary virtual window transparency color. The secondary virtual window is often referred to as being the graphics plane and may not be available on all platforms.

B.36.3 t_ve_callback_video_capture_device_query_s

The **t_ve_callback_video_capture_device_query_s** structure is used within the **t_ve_callback_device_query_u** union as defined in Appendix B.36 on Page 426. The **t_ve_callback_video_capture_device_query_s** structure allows for the storage of specific information from a queried video capture device. The structure is defined in **ve_callback.h** and is described below:

```
typedef struct{
  UWord32                                frame_rate;
  UWord32                                frame_buffer_address;
  UWord32                                image_width;
  UWord32                                image_height,
  t_ve_callback_transfermode_e           transfer_mode;
  t_ve_callback_video_color_format_e     color_format;
}t_ve_callback_video_capture_device_query_s;
```

The elements of this structure are as follows:

Element	Definition
frame_rate	The frame rate capability of the video capture device.
frame_buffer_address	The address of the video capture device buffer to be read from.
image_width	The horizontal size of the video image captured.
image_height	The vertical size of the video image captured.
transfer_mode	An enumerated type, **t_ve_callback_transfermode_e** as defined in Appendix B.177 on Page 599. The enumerated type determines whether the transfer process id to be via polling or be interrupt driven.
color_format	An enumerated type, **t_ve_callback_video_color_format_e** as defined in Appendix B.175 on Page 598. The enumerated type determines the color format supported by the video image capture device.

B.37 t_ve_callback_network_config_s

The **t_ve_callback_network_config_s** structure is used within the user written Data Manager interface function registered through the **VE_RegisterPacketTransportConfigure** function. The **t_ve_callback_network_config_s** structure allows for the configuration of the network interface. The structure is defined in **ve_callback.h** and is described below:

```
typedef struct{
  UWord8  tos;
}t_ve_callback_network_config_s;
```

The elements of the **t_ve_callback_network_config_s** structure are as follows:

Element	Definition
tos	The Type Of Service (TOS) to be used. The TOS can be as per the IP TOS (RFC0791 and RFC1349) or Differentiated Services TOS (RFC2474).

B.38 t_ve_statistic_u

The **t_ve_statistic_u** union is used by calls made to the **VE_GetStatistics** function. The encapsulated elements of this union allow for storage of the statistics requested by the user application. The **t_ve_statistic_u** union is defined in **ve_defs.h** and is described below:

```
typedef union{
  t_ve_stat_system_calls_s              system_calls;
  t_ve_stat_channel_video_encoder_s     channel_video_encoder;
  t_ve_stat_channel_video_decoder_s     channel_video_decoder;
  t_ve_stat_channel_rtcp_s              channel_rtcp;
  t_ve_stat_channel_rtcp_peer_s         channel_rtcp_peer;
  t_ve_stat_channel_av_sync_s           channel_av_sync;
}t_ve_statistic_u;
```

The union elements are as follows:

Element	Definition
system_calls	A structure, t_ve_stat_system_calls_s as described in Appendix B.38.1on Page 432, the provides statistics on the system calls.
channel_video_encoder	A structure, t_ve_stat_channel_video_encoder_s as described in Appendix B.38.3 on Page 433, the provides video encoder statistics on the specified channel.
channel_video_decoder	A structure, t_ve_stat_channel_video_decoder_s as described in Appendix B.38.4 on Page 434, the provides video decoder statistics on the specified channel.
channel_rtcp	A structure, t_ve_stat_channel_rtcp_s as described in Appendix B.38.5 on Page 435, the provides RTCP statistics on the specified channel.
channel_rtcp_peer	A structure, t_ve_stat_channel_rtcp_peer_s as described in Appendix B.38.6 on Page 438, the provides RTCP peer statistics on the specified channel.
channel_av_sync	A structure, t_ve_stat_channel_av_sync_s as described in Appendix B.38.2 on Page 432, the provides audio to video synchronization statistics on the specified channel.

B.38.1 t_ve_stat_system_calls_s

The **t_ve_stat_system_calls_s** structure is used within the **t_ve_statistic_u** union described in Appendix B.38 on Page 431. The **t_ve_stat_system_calls_s** structure is used to obtain the VeriCall Edge calls statistics, which are all valid from the last system start-up. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  UWord32    completed_calls;
  UWord32    busy_calls;
  UWord32    failed_calls;
}t_ve_stat_system_calls_s;
```

The elements of this structure are as follows:

Element	Definition
completed_calls	The number of completed calls since the VeriCall Edge system was last started.
busy_calls	The number of busy calls received since the VeriCall Edge system was last started.
failed_calls	The number of failed calls since the VeriCall Edge system was last started.

B.38.2 t_ve_stat_channel_av_sync_s

The **t_ve_stat_channel_av_sync_s** structure is used within the **t_ve_statistic_u** union described in Appendix B.38 on Page 431. The **t_ve_stat_channel_av_sync_s** structure is used to return the statistics for all the sub-channels of the channel identified from which to obtain the audio visual synchronization statistics. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_stat_av_sync_s sub_channel[VE_USER_SUB_CHANNEL_INDEX_COUNT];
}t_ve_stat_channel_av_sync_s;
```

The element of this structure is as follows:

Element	Definition
sub_channel	An array of structures, of type **t_ve_stat_av_sync_s** as described in Appendix B.38.2.1 on Page 433, used to return the audio visual synchronization statistics for all the sub-channels of the identified channel.

Statistics for ALL sub-channels will be returned. It is the responsibility of the user application to be aware of the number of sub-channels (primary only or primary and secondary) associated with the identified channel.

B.38.2.1 t_ve_stat_av_sync_s

The **t_ve_stat_av_sync_s** structure is used within the **t_ve_stat_channel_av_sync_s** structure described in Appendix B.38.2 on Page 432. The **t_ve_stat_av_sync_s** structure is used to provide the audio to visual synchronization statistics for the specified sub-channel. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  Word32                          ave_video_to_audio_diff_ms;
  t_ve_stat_av_sync_correlate_e   channel_av_is_correlated;
}t_ve_stat_av_sync_s;
```

The elements of this structure are as follows:

Element	Definition
ave_video_to_audio_diff_ms	The average video to audio synchronization difference in milliseconds.
channel_av_is_correlated	An enumerated type, **t_ve_stat_av_sync_correlate_e** as described in Appendix B.180 on Page 601, which determines whether correlation is enabled or not.

B.38.3 t_ve_stat_channel_video_encoder_s

The **t_ve_stat_channel_video_encoder_s** structure is used within the **t_ve_statistic_u** union described in Appendix B.38 on Page 431. The **t_ve_stat_channel_video_encoder_s** structure is used to return the statistics for all the sub-channels of the channel identified from which to obtain the video encoder statistics. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_stat_video_encoder_s
                    sub_channel[VE_USER_SUB_CHANNEL_INDEX_COUNT];
}t_ve_stat_channel_video_encoder_s;
```

The elements of this structure are as follows:

Element	Definition
sub_channel	An array of structures, **t_ve_stat_video_encoder_s** as defined in Appendix B.38.3.1 on Page 434, used to return the video encoder statistics for all the sub-channels of the identified channel.

Statistics for ALL sub-channels will be returned. It is the responsibility of the user application to be aware of the number of sub-channels (primary only or primary and secondary) associated with the identified channel.

B.38.3.1 t_ve_stat_video_encoder_s

The **t_ve_stat_video_encoder_s** structure is used within the **t_ve_stat_channel_video_encoder_s** structure described in Appendix B.38.3 on Page 433. The **t_ve_stat_video_encoder_s** structure is used to return the video encoder statistics for a sub-channel. These statistics are never cleared unless the statistics are reset. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  UWord32       frames_encoded;
  UWord32       frames_not_encoded;
}t_ve_stat_video_encoder_s;
```

The elements of this structure are as follows:

Element	Definition
frames_encoded	The number of video frames encoded.
frames_not_encoded	The number of video frames that have not been encoded.

B.38.4 t_ve_stat_channel_video_decoder_s

The **t_ve_stat_channel_video_decoder_s** structure is used within the **t_ve_statistic_u** union described in Appendix B.38 on Page 431. The **t_ve_stat_channel_video_decoder_s** structure is used to return the statistics for all the sub-channels of the channel identified from which to obtain the video decoder statistics. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_stat_video_decoder_s
                        sub_channel[VE_USER_SUB_CHANNEL_INDEX_COUNT];
}t_ve_stat_channel_video_decoder_s;
```

The elements of this structure are as follows:

Element	Definition
sub_channel	An array of structures, **t_ve_stat_video_decoder_s** as defined in Appendix B.38.4.1 on Page 435, used to return the video decoder statistics for all the sub-channels of the identified channel.

Statistics for ALL sub-channels will be returned. It is the responsibility of the user application to be aware of the number of sub-channels (primary only or primary and secondary) associated with the identified channel.

B.38.4.1 t_ve_stat_video_decoder_s

The **t_ve_stat_video_decoder_s** structure is used within the **t_ve_stat_channel_video_decoder_s** structure described in Appendix B.38.4 on Page 434. The **t_ve_stat_video_decoder_s** structure is used to return the video decoder statistics for a sub-channel. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  UWord32    frames_decoded;
  UWord32    frames_not_decoded;
}t_ve_stat_video_decoder_s;
```

The elements of this structure are as follows:

Element	Definition
frames_decoded	The number of video frames decoded.
frames_not_decoded	The number of video frames that have not been decoded (dropped).

B.38.5 t_ve_stat_channel_rtcp_s

The **t_ve_stat_channel_rtcp_s** structure is used within the **t_ve_statistic_u** union described in Appendix B.38 on Page 431. The **t_ve_stat_channel_rtcp_s** structure is used to return the statistics for all the sub-channels of the channel identified from which to obtain the RTCP statistics. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_stat_rtcp_s
                    sub_channel[VE_USER_SUB_CHANNEL_INDEX_COUNT];
}t_ve_stat_channel_rtcp_s;
```

The elements of this structure are as follows:

Element	Definition
sub_channel	An array of structures, **t_ve_stat_rtcp_s** as defined in Appendix B.38.5.1 on Page 436, used to return the RTCP statistics for all the sub-channels of the identified channel.

Statistics for ALL sub-channels will be returned. It is the responsibility of the user application to be aware of the number of sub-channels (primary only or primary and secondary) associated with the identified channel.

B.38.5.1 t_ve_stat_rtcp_s

The **t_ve_stat_rtcp_s** structure is used within the **t_ve_stat_channel_rtcp_s** structure described in Appendix B.38.5 on Page 435. The **t_ve_stat_rtcp_s** structure is used to return the RTCP statistics for a sub-channel. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  UWord32               rtcp_count;
  t_ve_media_type_e     media_type;
  UWord32               sr_ssrc;
  UWord32               sr_ntp_timestamp_msb;
  UWord32               sr_ntp_timestamp_lsb;
  UWord32               sr_rtp_timestamp;
  UWord32               sr_packet_count;
  UWord32               sr_octet_count;
  UWord32               rr_ssrc;
  UWord32               rr_lost_fraction;
  UWord32               rr_accumulated_packets_lost;
  UWord32               rr_highest_seq_number;
  UWord32               rr_seq_number_cycles;
  UWord32               rr_inter_arrival_jitter;
  UWord32               rr_last_sr;
  UWord32               rr_delay_since_last_sr;
  UWord32               recv_packet_count;
  UWord32               recv_octet_count;
  UWord32               calc_round_trip_time;
  UWord32               average_inter_arrival_jitter;
  UWord32               average_round_trip_time;
}t_ve_stat_rtcp_s;
```

Continued >>

The elements of this structure are as follows:

Element	Definition
rtcp_count	The received rtcp packet count since stream started.
media_type	The enumerated media type, **t_ve_media_type_e** as described in Appendix B.106 on Page 537.
sr_ssrc	The sender report SSRC.
sr_ntp_timestamp_msb	The sender report NTP timestamp. This is the most significant byte of a 64-bit word.
sr_ntp_timestamp_lsb	The sender report NTP timestamp. This is the least significant byte of a 64-bit word.
sr_rtp_timestamp	The sender report RTP timestamp.
sr_packet_count	The sender report packet count.
sr_octet_count	The sender report octet count.
rr_ssrc	The receiver report SSRC.
rr_lost_fraction	The number of receiver report lost fractions.
rr_accumulated_packets_lost	The number of accumulated receiver report lost packets.
rr_highest_seq_number	The highest receiver report sequence number.
rr_seq_number_cycles	The number of receiver report sequence cycles.
rr_inter_arrival_jitter	The receiver report inter arrival jitter.
rr_last_sr	The time between this receiver report and the last sender report.
rr_delay_since_last_sr	The delay between the receiver report and the last sender report.
recv_packet_count	The number of received packets.
recv_octet_count	The number of received octets.
calc_round_trip_time	The calculated round-trip time.
average_inter_arrival_jitter	The average inter arrival jitter since stream started.
average_round_trip_time	The average round trip time since stream started.

B.38.6 t_ve_stat_channel_rtcp_peer_s

The **t_ve_stat_channel_rtcp_peer_s** structure is used within the **t_ve_statistic_u** union described in Appendix B.38 on Page 431. The **t_ve_stat_channel_rtcp_peer_s** structure is used to return the statistics for all the sub-channels of the channel identified from which to obtain the RTCP peer statistics. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_stat_rtcp_peer_s
                    sub_channel[VE_USER_SUB_CHANNEL_INDEX_COUNT];
}t_ve_stat_channel_rtcp_peer_s;
```

The elements of this structure are as follows:

Element	Definition
sub_channel	An array of structures, **t_ve_stat_video_encoder_s** as defined in Appendix B.38.6.1 on Page 439, used to return the RTCP peer statistics for all the sub-channels of the identified channel.

Statistics for ALL sub-channels will be returned. It is the responsibility of the user application to be aware of the number of sub-channels (primary only or primary and secondary) associated with the identified channel.

B.38.6.1 t_ve_stat_video_encoder_s

The **t_ve_stat_video_encoder_s** structure is used within the **t_ve_stat_channel_rtcp_peer_s** structure described in Appendix B.38.6 on Page 438. The **t_ve_stat_video_encoder_s** structure is used to return the video encoder statistics for a sub-channel. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  UWord32       sr_ssrc;
  UWord32       sr_ntp_timestamp_msb;
  UWord32       sr_ntp_timestamp_lsb;
  UWord32       sr_rtp_timestamp;
  UWord32       sr_packet_count;
  UWord32       sr_octet_count;
  UWord32       rr_ssrc;
  UWord32       rr_lost_fraction;
  UWord32       rr_accumulated_packets_lost;
  UWord32       rr_highest_seq_number;
  UWord32       rr_seq_number_cycles;
  UWord32       rr_inter_arrival_jitter;
  UWord32       rr_last_sr;
  UWord32       rr_delay_since_last_sr;
}t_ve_stat_rtcp_peer_s;
```

The elements of this structure are as follows:

Element	Definition
sr_ssrc	The sender report SSRC.
sr_ntp_timestamp_msb	The sender report NTP timestamp. This is the most significant byte of a 64-bit word.
sr_ntp_timestamp_lsb	The sender report NTP timestamp. This is the least significant byte of a 64-bit word.
sr_rtp_timestamp	The sender report RTP timestamp.
sr_packet_count	The sender report packet count.
sr_octet_count	The sender report octet count.
rr_ssrc	The receiver report SSRC.
rr_lost_fraction	The number of receiver report lost fractions.
rr_accumulated_packets_lost	The number of accumulated receiver report lost packets.
rr_highest_seq_number	The highest receiver report sequence number.
rr_seq_number_cycles	The number of receiver report sequence cycles.
rr_inter_arrival_jitter	The receiver report inter arrival jitter.
rr_last_sr	The time between this receiver report and the last sender report.
rr_delay_since_last_sr	The delay between the receiver report and the last sender report.

B.39 t_ve_sip_user_registration_s

The **t_ve_sip_user_registration_s** structure is used in calls made to the **VE_SIPAddUsersRegistrationData** and **VE_SIPUpdateUsersRegistrationData** functions. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The structure allows for registration details of a user to be added to the VeriCall Edge system, which will also be used if the user is to be optionally registered with a SIP registration server. The **t_ve_sip_user_registration_s** structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
  t_ve_sip_user_uri_s               sip_user_uri;
  t_ve_sip_contact_info_s           sip_contact_info;
  t_ve_sip_registration_server_s    sip_reg_server;
}t_ve_sip_user_registration_s;
```

The elements of the **t_ve_sip_user_registration_s** structure are as follows:

Element	Definition
sip_user_uri	A structure, **t_ve_sip_user_uri_s** as defined in Appendix B.39.1 on Page 440, which defines the SIP URI of this user.
sip_contact_info	A structure, **t_ve_sip_contact_info_s,** as defined in Appendix B.39.2 on Page 441, which defines the optional contact information of this user.
sip_reg_server	A structure, **t_ve_sip_registration_server_s**, as defined in Appendix B.39.3 on Page 442, which defines the SIP registration server against which this user may be registered with.

B.39.1 t_ve_sip_user_uri_s

The **t_ve_sip_user_uri_s** structure is used within the **t_ve_sip_user_registration_s** structure as defined in Appendix B.39 on Page 440 and the **t_ve_sip_call_id_s** structure as defined in Appendix B.33.32.1 on Page 390. The **t_ve_sip_user_uri_s** structure defines the URI of the user, either the user to be called or the calling user. This structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
  Word8      display_name[VE_MAX_URI_DISPLAY_NAME_STRING];
  Word8      uri[VE_MAX_URI_STRING_SIZE];
}t_ve_sip_user_uri_s;
```

The elements of this structure are as follows:

Element	Definition
display_name	The optional NULL terminated name string displayed by the SIP client. Because this is an optional element of this structure, this element may be NULL.
uri	A valid NULL terminated SIP URI string consisting of the user name and hostname.

B.39.2 t_ve_sip_contact_info_s

The **t_ve_sip_contact_info_s** structure is part of the **t_ve_user_registration_s** structure defined in Appendix B.39 on Page 440 and the **t_ve_sip_call_id_s** structure as defined in Appendix B.33.32.1 on Page 390. The **t_ve_sip_contact_info_s** structure is used to configure the SIP user contact information with the VeriCall Edge system. This structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
  UWord16    num_contacts;
  Word8      contacts[VE_MAX_NUM_CONTACTS][VE_MAX_URI_STRING_SIZE];
}t_ve_sip_contact_info_s;
```

The elements of this structure are as follows:

Element	Definition
num_contacts	The number of contacts associated with this user.
Contacts	Multi-dimensional array holding the contact identifier and contact details of the user.

It is a requirement to have at least one contact, if the user application does not complete a contact entry, a default will automatically be used.

B.39.3 t_ve_sip_registration_server_s

The **t_ve_sip_registration_server_s** structure is used within the **t_ve_sip_user_registration_s** structure as defined in Appendix B.39 on Page 440. The **t_ve_sip_registration_server_s** structure defines the SIP registration server against which the user is to be registered. This structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
  Word8
     authentication_username[VE_MAX_AUTHENTICATION_USERNAME_SIZE];
  Word8   authentication_realm[VE_MAX_AUTHENTICATION_REALM_SIZE];
  Word8   password[VE_MAX_PASSWORD_SIZE];
  Word8   address[VE_MAX_URI_STRING_SIZE];
  t_ve_sip_transport_protocol_e   transport_protocol;
}t_ve_sip_registration_server_s;
```

The elements of this structure are as follows:

Element	Definition
authentication_username	The user name used for authentication. This is only needed if it differs from the username in the SIP URI. If it is set to `NULL` then the user name from the URI is used
authentication_realm	Reserved for future use.
password	Optional password, which can be `NULL`.
address	The IPv4 or FQDN address of the SIP registration server.
transport_protocol	An enumerated type, **t_ve_sip_transport_protocol_e** as described in Appendix B.203 on Page 623, which allows the SIP registration transport type to be defined.

If the VeriCall Edge system is to be used without a SIP Registration Server, the *address* field of this structure should be zeroed and VE_SIPRegisterUser should not be called.

The VeriCall Edge system uses the network byte ordering for IP addresses when a UWord32 value is required.

B.40 t_ve_sip_stack_liberalisation_s

The **t_ve_sip_stack_liberalisation_s** structure is used within the **t_ve_sip_default_configuration_s** structure which is described in Appendix B.17 on Page 284. The structure is currently reserved for future use. The **t_ve_sip_stack_liberalisation_s** structure is defined in **ve_sip_configuration.h** and is described below:

```
typedef struct{
  t_ve_sip_state_flag_default_configuration_e  reserved;
}t_ve_sip_stack_liberalisation_s;
```

The element of this structure is as follows:

Element	Definition
reserved	This element is reserved for future use.

B.41 t_ve_sip_stack_feature_support_s

The **t_ve_sip_stack_feature_support_s** structure used within the **t_ve_sip_default_configuration_s** structure which is described in Appendix B.17 on Page 284. The structure allows for the configuration of some of the VeriCall Edge SIP stack features. The **t_ve_sip_stack_feature_support_s** structure is defined in **ve_sip_configuration.h** and is described below:

```
typedef struct{
  t_ve_sip_state_flag_default_configuration_e  use_short_headers;
  t_ve_sip_state_flag_default_configuration_e  use_rport;
  UWord32                                     register_keep_alive;
  t_ve_sip_state_flag_default_configuration_e  include_date_header;
}t_ve_sip_stack_feature_support_s;
```

Continued >>

The elements of this structure are as follows:

Element	Definition
use_short_headers	Am enumeration type, **t_ve_sip_state_flag_default_configuration_e** as described in Appendix B.183 on Page 605, which is a flag to enable/disable using short headers for transmitted messages. Enabling this flag will, for example, change the "From:" header to "f:". The SIP stack will always be able to receive short headers regardless of this configuration option. By default, reception of short headers is disabled.
use_rport	Am enumeration type, **t_ve_sip_state_flag_default_configuration_e** as described in Appendix B.183 on Page 605, which is a flag to enable/disable rport support. By default this is enabled.
register_keep_alive	A control word for using REGISTER messages to perform keep-alive for connection between VeriCall Edge and the server. This is required if NAT sits between them. • To disable this feature a value of 0x00 should be used. • To enable this feature a value greater than 0x00 should be used where the value also has the additional capability of specifying the percentage of the NAT binding lifetime that this keep-alive should run at. It is recommended that this be greater than 50%, to keep the overhead reasonable, and no greater than 80% to ensure connection is kept alive. • By default this capability is disabled, value 0x00.
include_date_header	An enumeration type, **t_ve_sip_state_flag_default_configuration_e** as described in Appendix B.183 on Page 605, which is a flag to enable/disable the date header. If enabled the date header will be included in all SIP headers. Note: for the date to be correct the operating system must have the date and time set correctly. By default this is enabled.

B.42 t_ve_sip_routing_s

The **t_ve_sip_routing_s** structure used within the **t_ve_sip_default_configuration_s** structure which is described in Appendix B.17 on Page 284. The structure allows for the configuration of network routes and proxies to be defined. The **t_ve_sip_routing_s** structure is defined in **ve_sip_configuration.h** and is described below:

```
typedef struct{
 t_ve_sip_state_flag_default_configuration_e  auto_proxy_switch;
 t_ve_sip_routing_data_s routing_data[VE_SIP_MAX_NUM_ROUTE_DATA];
 UWord32                num_routes;
}t_ve_sip_routing_s;
```

The elements of this structure are as follows:

Element	Definition
auto_proxy_switch	An enumeration type, **t_ve_sip_state_flag_default_configuration_e** as described in Appendix B.183 on Page 605, which defines whether the SIP stack should automatically change to a new proxy if the current one fails. The SIP stack will try all proxies up to the number define by *num_routes*. By default his is enabled.
routing_data	An array of structures, **t_ve_sip_routing_data_s** as described in Appendix B.42.1 on Page 445, which defines a number of routes outbound proxies. The maximum number of routes/proxies definable cannot exceed VE_SIP_MAX_NUM_ROUTE_DATA.
num_routes	The number of routes defined in the *routing_data* array of structures.

B.42.1 t_ve_sip_routing_data_s

The **t_ve_sip_routing_data_s** structure used within the **t_ve_sip_routing_s** structure which is described in Appendix B.42 on Page 445. The structure allows for the configuration of a network route and proxy to be defined. The **t_ve_sip_routing_data_s** structure is defined in **ve_sip_configuration.h** and is described below:

```
typedef struct{
  Word8      route_set[VE_SIP_MAX_ROUTE_SET_STRING_SIZE];
  Word8      proxy[VE_SIP_MAX_PROXY_STRING_SIZE];
  t_ve_sip_transport_protocol_e        transport_protocol;
}t_ve_sip_routing_data_s;
```

Continued >>

The elements of this structure are as follows:

Element	Definition
route_set	Allows the definition of a pre-existing route. The route set is associated with the proxy, e.g. sip:route1, sip:route2. The maximum string to describe the route must not exceed `VE_SIP_MAX_ROUTE_SET_STRING_SIZE`, including the termination character. There are no routes set be default
proxy	Allows the definition of an outbound proxy, e.g. proxy1.trinityconvergence.com. The maximum string to describe the proxy must not exceed `VE_SIP_MAX_PROXY_STRING_SIZE`, including the termination string. There are no proxies set by default.
transport_protocol	An enumeration type, **t_ve_sip_transport_protocol_e** as described in Appendix B.203 on Page 623, that defines the transport protocol to be used. The default is UDP.

B.43 t_ve_sip_date_s

The **t_ve_sip_date_s** structure used within the **t_ve_sip_call_id_s** structure which is described in Appendix B.33.32.1 on Page 390. The structure contains the SIP date header information, which may or may not be enabled. The **t_ve_sip_date_s** structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
  t_ve_sip_date_state_e     state;
  Word8         weekday[VE_SIP_MAX_DATE_WEEKDAY_STRING_SIZE];
  UWord16       day;
  Word8         month[VE_SIP_MAX_DATE_MONTH_STRING_SIZE];
  UWord32       year;
  UWord16       hour;
  UWord16       minute;
  UWord16       second;
}t_ve_sip_date_s;
```

The elements of this structure are as follows:

Element	Definition
state	An enumeration type, **t_ve_sip_date_state_e** as described in Appendix B.186 on Page 607, which defines whether the use of SIP date header is to be active or inactive.
weekday	This can be one of the following strings: • "Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"
day	This can be a number in the range of 1 to 31.
month	This can be one of the following strings: • "Jan", "Feb", "Mar", "Apr", "May", "Jun", "Jul", "Aug", "Sep", "Oct", "Nov", "Dec"
year	The year is a 4 digit number.
hour	This can be a number in the range of 0 to 23.
minute	This can be a number in the range of 0 to 59.
second	This can be a number in the range of 0 to 59.

B.44 t_ve_sip_tx_session_response_data_s

The **t_ve_sip_tx_session_response_data_s** structure used within calls to the **VE_SIPSessionResponse** function. The structure contains the SIP session response content data. The **t_ve_sip_tx_session_response_data_s** structure is defined in **ve_sip_defs.h** and described below:

```
typedef struct{
  t_ve_sip_contact_info_s                    sip_contact_info;
  t_ve_sip_tx_session_response_content_e     content_type;
  t_ve_sip_tx_session_response_content_u     content;
  t_ve_sip_require_prack_e                   require_prack;
  Word8        alert_info[VE_SIP_MAX_ALERT_INFO_STRING_SIZE];
}t_ve_sip_tx_session_response_data_s;

typedef union{
  t_ve_stream_configuration_s     ve_stream_config;
  t_ve_sip_content_data_s         content_data;
}t_ve_sip_tx_session_response_content_u;
```

The elements of this structure are as follows:

Element	Definition
sip_contact_info	A structure, **t_ve_sip_contact_info_s** as described in Appendix B.39.2 on Page 441, which contains extra contact information which can be sent in the response. This information is also used for call forwarding for SIP responses in the 3xx range.
content_type	An enumeration type, **t_ve_sip_authentication_required_e** as described in Appendix B.188 on Page 608, which allows control of when SDP content is included in a response.
content	A union of structures, whose selection is determined by the *content_type* element, which allows the SDP content to be included in the response. The selected structure can be one of the following: **t_ve_stream_configuration_s** as described in Appendix B.24 on Page 298. **t_ve_sip_content_data_s** as described in Appendix B.45.2 on Page 450.
require_prack	An enumeration type, **t_ve_sip_date_state_e** as described in Appendix B.186 on Page 607, which is a flag to inform the destination that PRACK is required to be used. This flag will only effect provisional responses. Note that the required header will only be added if the destination supports the feature.
alert_info	Populates the Alert-Info SIP header. If this is NULL then no Alert-Info is included. Alert-Info will only be added to a 180 Ringing response.

B.45 t_ve_sip_rx_session_response_data_s

The **t_ve_sip_rx_session_response_data_s** structure used within the following structures:

- **t_ve_user_msg_sip_invite_response_rx_s** described in Appendix B.33.10 on Page 367.
- **t_ve_user_msg_sip_update_response_rx_s** described in Appendix B.33.12 on Page 369.
- **t_ve_user_msg_sip_reinvite_response_rx_s** described in Appendix B.33.13 on Page 370.

The structure contains the SIP session response content data and is passed to the user application. The **t_ve_sip_rx_session_response_data_s** structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
  t_ve_sip_rx_session_response_content_e    content_type;
  t_ve_sip_rx_session_response_content_u    content;
  t_ve_sip_authentication_data_s            authentication_data;
}t_ve_sip_rx_session_response_data_s;

typedef union{
  t_ve_stream_configuration_s     ve_stream_config;
  t_ve_sip_content_data_s         content_data;
}t_ve_sip_rx_session_response_content_u;
```

The elements of this structure are as follows:

Element	Definition
content_type	An enumeration type, **t_ve_sip_call_hold_method_e** as described in Appendix B.190 on Page 609, which indicates if the received response contains content data.
content	A union of structures, whose selection is determined by the *content_type* element, which allows the Session Description Protocol (SDP) content to be included in the response. The selected structure can be one of the following: **t_ve_stream_configuration_s** as described in Appendix B.24 on Page 298. **t_ve_sip_content_data_s** as described in Appendix B.45.2 on Page 450.
authentication_data	A structure, **t_ve_sip_authentication_data_s** as described in Appendix B.45.1 on Page 450, which is used when the destination requires authentication. This allows the user application to challenge the user for a password and then retry with the supplied password.

B.45.1 t_ve_sip_authentication_data_s

The **t_ve_sip_authentication_data_s** structure used within the **t_ve_sip_rx_session_response_data_s** structure which is described in Appendix B.45 on Page 449. The structure contains the authentication realm as well as allowing determination of whether authentication is required or not. The **t_ve_sip_authentication_data_s** structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
  t_ve_sip_authentication_required_e      authentication_required;
  Word8   authentication_realm[VE_MAX_AUTHENTICATION_REALM_SIZE];
}t_ve_sip_authentication_data_s;
```

The elements of this structure are as follows:

Element	Definition
authentication_required	An enumeration type, **t_ve_sip_authentication_required_e** as described in Appendix B.188 on Page 608, which is a flag that determine whether authentication is required. If no authentication is required, the remainder of this structure will not be set.
authentication_realm	The realm in the authentication challenge.

B.45.2 t_ve_sip_content_data_s

The **t_ve_sip_content_data_s** structure used as an argument to the **VE_SIPSendMessage** and **VE_SIPSendInfo** functions as well as the following structures:

- **t_ve_sip_tx_session_response_data_s** described in Appendix B.44 on Page 448.
- **t_ve_sip_rx_session_response_data_s** described in Appendix B.45 on Page 449.
- **t_ve_user_msg_sip_message_rx_s** described in Appendix B.33.18 on Page 375.
- **t_ve_user_msg_sip_message_response_rx_s** described in Appendix B.33.19 on Page 376.
- **t_ve_user_msg_sip_info_rx_s** described in Appendix B.33.22 on Page 379.
- **t_ve_user_msg_sip_info_response_rx_s** described in Appendix B.33.23 on Page 380.

Continued >>

The structure contains the SIP content data and the settings of that data. The **t_ve_sip_content_data_s** structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
Word8
content_type[VE_SIP_MAX_CONTENT_DATA_CONTENT_TYPE_LEN];
Word8
content_encoding[VE_SIP_MAX_CONTENT_DATA_CONTENT_ENCODING_LEN];
Word8
content_language[VE_SIP_MAX_CONTENT_DATA_CONTENT_LANGUAGE_LEN];
Word8
content_disposition[VE_SIP_MAX_CONTENT_DATA_CONTENT_DISPOSITION_LEN];
UWord32        content_length;
Word8          content[VE_SIP_MAX_CONTENT_DATA_CONTENT_LEN];
UWord16        part_number;
UWord16        number_of_parts;
}t_ve_sip_content_data_s;
```

The elements of this structure are as follows:

Element	Definition
content_type	A recognized string that determines the SIP content data type, for example “text/plain”.
content_encoding	An optional field that contains a recognized string that determines the encoding method used, for example “gzip”.
content_language	An optional field that contains a recognized string that determines the content language, for example “en”.
content_disposition	An optional field that contains a recognized string that determines the content disposition, for example “session”.
content_length	The content length.
content	The content data, whose length is determined by the *content_length* element.
part_number	Reserved for future use.
number_of_parts	Reserved for future use.

B.46 t_ve_sip_presence_data_s

The **t_ve_sip_presence_data_s** structure used within the **VE_SIPAddUsersRegistrationData**, **VE_SIPUpdateUsersRegistrationData** and **VE_SIPGetUsersRegistrationData** API functions as well as within the **t_ve_user_msg_sip_presence_notify_rx_s** structure, described in Appendix B.33.28 on Page 385. The structure contains the SIP presence data, if available. The **t_ve_sip_presence_data_s** structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
  t_ve_sip_presence_type_e     type;
  t_ve_sip_presence_data_u     data;
}t_ve_sip_presence_data_s;

typedef union{
  t_ve_sip_presence_xml_pidf_presence_s     xml_pidf;
}t_ve_sip_presence_data_u;
```

The elements of this structure are as follows:

Element	Definition
type	An enumeration type, **t_ve_sip_presence_type_e** as described in Appendix B.195 on Page 615, which indicates the data type of the SIP presence data.
data	A union of structures, whose selection is determined by the *type* element, which allows the SIP presence data to be presented to the user application. The selected structure can be one of the following: **t_ve_sip_presence_xml_pidf_presence_s** as described in Appendix B.46.1 on Page 453.

B.46.1 t_ve_sip_presence_xml_pidf_presence_s

The **t_ve_sip_presence_xml_pidf_presence_s** structure used within the **t_ve_sip_presence_data_s** structure which is described in Appendix B.46 on Page 452. The structure contains the SIP presence information. The **t_ve_sip_presence_xml_pidf_presence_s** structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
  Word8                                entity[VE_MAX_URI_STRING_SIZE];
  UWord16                              num_tuples;
  t_ve_sip_presence_xml_pidf_tuple_s   tuple[VE_SIP_MAX_NUM_TUPLES];
  UWord16                              num_notes;
  t_ve_sip_presence_xml_pidf_note_s    note[VE_SIP_MAX_NUM_NOTES];
}t_ve_sip_presence_xml_pidf_presence_s;
```

The elements of this structure are as follows:

Element	Definition
entity	The entity presence information. This string cannot exceed `VE_MAX_URI_STRING_SIZE` in length, including the termination character.
num_tuples	The number of TUPLES present.
tuple	An array of structures, **t_ve_sip_presence_xml_pidf_tuple_s** as described in B.46.1.1 on Page 454, which contains the TUPLE information of the SIP presence.
num_notes	The number of NOTES present.
note	An array of structures, **t_ve_sip_presence_xml_pidf_note_s** as described in B.46.1.4 on Page 456, which contains the NOTE information of the SIP presence.

B.46.1.1 t_ve_sip_presence_xml_pidf_tuple_s

The **t_ve_sip_presence_xml_pidf_tuple_s** structure used within the **t_ve_sip_presence_xml_pidf_presence_s** structure which is described in Appendix B.46.1 on Page 453. The structure contains information on the status and information on the SIP presence TUPLE. The **t_ve_sip_presence_xml_pidf_tuple_s** structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
  Word8      id[VE_SIP_MAX_ID_LEN];
  t_ve_sip_presence_xml_pidf_status_s     status;
  t_ve_sip_presence_xml_pidf_contact_s    contact;
  t_ve_sip_presence_xml_pidf_note_s       note;
  Word8      timestamp[VE_SIP_MAX_TIMESTAMP_LEN];
}t_ve_sip_presence_xml_pidf_tuple_s;
```

The elements of this structure are as follows:

Element	Definition
id	The identifier for this TUPLE. The identifier must not exceed `VE_SIP_MAX_ID_LEN` in length, including the termination character.
status	A structure, **t_ve_sip_presence_xml_pidf_status_s** as described in Appendix B.46.1.2 on Page 455, that determines the status of this TUPLE.
contact	A structure, **t_ve_sip_presence_xml_pidf_contact_s** as described in Appendix B.46.1.3 on Page 456, that contains the contact details of this TUPLE.
note	A structure, **t_ve_sip_presence_xml_pidf_note_s** as described in Appendix B.46.1.4 on Page 456, that contains the note of this TUPLE.
timestamp	The timestamp of this TUPLE, which cannot exceed `VE_SIP_MAX_TIMESTAMP_LEN` including the termination character.

B.46.1.2 t_ve_sip_presence_xml_pidf_status_s

The **t_ve_sip_presence_xml_pidf_status_s** structure used within the **t_ve_sip_presence_xml_pidf_presence_s** structure which is described in Appendix B.46.1 on Page 453. The structure contains the status of the basic SIP presence data. The **t_ve_sip_presence_xml_pidf_status_s** structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
  t_ve_sip_presence_xml_pidf_status_basic_type_e   basic;
}t_ve_sip_presence_xml_pidf_status_s;
```

The element of this structure is as follows:

Element	Definition
basic	An enumeration type, **t_ve_sip_presence_xml_pidf_status_basic_type_e** as described in Appendix B.192 on Page 610, which is a flag that determines the status of the basic presence data.

B.46.1.3 t_ve_sip_presence_xml_pidf_contact_s

The **t_ve_sip_presence_xml_pidf_contact_s** structure used within the **t_ve_sip_presence_xml_pidf_tuple_s** structure which is described in Appendix B.46.1.1 on Page 454. The structure contains the priority and data of the SIP presence contact. The **t_ve_sip_presence_xml_pidf_contact_s** structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
  UWord16      priority;
  Word8        value[VE_MAX_URI_STRING_SIZE];
}t_ve_sip_presence_xml_pidf_contact_s;
```

The elements of this structure are as follows:

Element	Definition
priority	The priority of this contact.
value	The contact information which cannot exceed `VE_MAX_URI_STRING_SIZE` in length, including the termination character.

B.46.1.4 t_ve_sip_presence_xml_pidf_note_s

The **t_ve_sip_presence_xml_pidf_note_s** structure used within the **t_ve_sip_presence_xml_pidf_presence_s** structure, described in Appendix B.46.1 on Page 453, and the **t_ve_sip_presence_xml_pidf_tuple_s** structure, described in Appendix B.46.1.1 on Page 454. The structure contains the language and note data of the SIP presence. The **t_ve_sip_presence_xml_pidf_note_s** structure is defined in ve_**sip_defs.h** and is described below:

```
typedef struct{
  Word8      language[VE_SIP_MAX_LANGUAGE_LEN];
  Word8      value[VE_SIP_MAX_NOTE_LEN];
}t_ve_sip_presence_xml_pidf_note_s;
```

The elements of this structure are as follows:

Element	Definition
priority	String that determines the language of the note which cannot exceed `VE_SIP_MAX_LANGUAGE_LEN` in length, including the termination character.
value	The note data which cannot exceed `VE_MAX_NOTE_LEN` in length, including the termination character.

B.47 t_ve_sip_mw_data_s

The **t_ve_sip_mw_data_s** structure used within the **t_ve_user_msg_sip_mw_notify_rx_s** structure which is described in Appendix B.33.30 on Page 387. The structure contains the SIP Message Waiting (MW) summary counts and data. The **t_ve_sip_mw_data_s** structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
  t_ve_sip_mw_indicatorstatus_e     mwi;
  UWord8                message_account[VE_MAX_URI_STRING_SIZE];
  UWord16               num_message_summaries;
  t_ve_sip_mw_msg_summary_data_s
                        message_summaries[VE_MAX_NUM_MW_SUMMARIES];
}t_ve_sip_mw_data_s;
```

The elements of this structure are as follows:

Element	Definition
mwi	An enumerated type, **t_ve_sip_mw_indicatorstatus_e** as described in Appendix B.197 on Page 617, which is a flag that either sets or clears the Message Waiting Indicator.
message_account	A string representation of the SIP URI of the message account summarized in this data. This string cannot exceed `VE_MAX_URI_STRING_SIZE` in length, including the termination character.
num_message_summaries	The number of summary messages given in the *`message_summaries`* array of structures.
message_summaries	An array of structures, **t_ve_sip_mw_msg_summary_data_s** as described in Appendix B.47.1 on Page 458, which contains the message summaries. The maximum number of message summaries available cannot exceed `VE_MAX_NUM_MW_SUMMARIES`.

B.47.1 t_ve_sip_mw_msg_summary_data_s

The **t_ve_sip_mw_msg_summary_data_s** structure used within the **t_ve_sip_mw_data_s** structure which is described in Appendix B.47 on Page 457. The structure contains the SIP Message Waiting (MW) summary. The **t_ve_sip_mw_msg_summary_data_s** structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
  t_ve_sip_mw_message_context_e   summary_context;
  UWord32                         new_messages;
  UWord32                         old_messages;
  UWord32                         new_urgent_messages;
  UWord32                         old_urgent_messages;
}t_ve_sip_mw_msg_summary_data_s;
```

The elements of this structure are as follows:

Element	Definition
summary_context	An enumeration type, **t_ve_sip_mw_message_context_e** as described in Appendix B.198 on Page 618, that determines the message waiting type this summary refer to.
new_messages	The number new messages available.
old_messages	The number of old messages still available.
new_urgent_messages	The number of new urgent messages available.
old_urgent_messages	The number of old urgent messages still available.

B.48 t_ve_sip_dialog_data_s

The **t_ve_sip_dialog_data_s** structure is part of the **t_ve_user_msg_sip_dialog_notify_rx_s** structure as described in Appendix B.33.34 on Page 392. The structure provides details of the destination SIP dialogs. The **t_ve_sip_dialog_data_s** structure is defined in **ve_sip_defs.h** and described below.

```
typedef struct
{
  UWord32              version;
  Word8                entity[VE_MAX_URI_STRING_SIZE];
  UWord32              num_dialogs;
  t_ve_sip_dialog_s    dialog[VE_SIP_MAX_NUM_DIALOGS];

}t_ve_sip_dialog_data_s;
```

The elements of this structure are as follows:

Element	Definition
version	The version number of the dialog data. Increments by one each time the structure is updated by a received NOTIFY request.
entity	The URI of the user that the dialog information relates to.
num_dialogs	The number of dialogs in the dialog array up to a maximum of VE_SIP_MAX_NUM_DIALOGS.
dialog	An array of structures, **t_ve_sip_dialog_s** as defined in Appendix B.48.1 on Page 460, that identifies the dialogs in the received NOTIFY request.

B.48.1 t_ve_sip_dialog_s

The **t_ve_sip_dialog_s** structure is part of the **t_ve_sip_dialog_data_s** structure described in Appendix B.48 on Page 459. The structure contains information for a particular SIP dialog. The **t_ve_sip_dialog_s** structure is defined in **ve_sip_defs.h** and described below.

```
typedef struct{
  t_ve_sip_dialog_id_s          dialog_id;
  t_ve_sip_dialog_state_s       dialog_state;
  t_ve_sip_dialog_direction_s   dialog_direction;
  t_ve_sip_dialog_duration_s    dialog_duration;
}t_ve_sip_dialog_s;
```

The elements of this structure are as follows:

Element	Definition
dialog_id	A structure, **t_ve_sip_dialog_id_s** as defined in Appendix B.48.1.1 on Page 460, that identifies the dialog.
dialog_state	A structure, **t_ve_sip_dialog_state_s** as defined in Appendix B.48.1.2 on Page 461, that describes the state of dialog.
dialog_direction	A structure**, t_ve_sip_dialog_direction_s** as defined in Appendix B.48.1.3 on Page 462, that identifies the direction of the dialog.
dialog_duration	A structure, **t_ve_sip_dialog_duration_s** as defined in Appendix B.48.1.4 on Page 462, that identifies the duration of the dialog.

B.48.1.1 t_ve_sip_dialog_id_s

The **t_ve_sip_dialog_id_s** structure is part of the **t_ve_sip_dialog_s** structure described in Appendix B.48.1 on Page 460. The structure contains information that identifies a SIP dialog. The **t_ve_sip_dialog_id_s** structure is defined in **ve_sip_defs.h** and described below.

```
typedef struct
{
  Word8      id[VE_SIP_MAX_DIALOG_ID_LEN];
  Word8      call_id[VE_SIP_MAX_CALL_ID_LEN];
  Word8      local_tag[VE_SIP_MAX_LOCAL_TAG_LEN];
  Word8      remote_tag[VE_SIP_MAX_REMOTE_TAG_LEN];
}t_ve_sip_dialog_id_s;
```

The elements of this structure are as follows:

Element	Definition
id	A random string that uniquely identifies the dialog.
call_id	The Call-ID header that forms part of the dialog.
local_tag	The tag field from the To header that forms part of the dialog.
remote_tag	The tag field from the From header that forms part of the dialog.

B.48.1.2 t_ve_sip_dialog_state_s

The **t_ve_sip_dialog_state_s** structure is part of the **t_ve_sip_dialog_s** structure described in Appendix B.48.1 on Page 460. The structure contains information about the state of a SIP dialog. The **t_ve_sip_dialog_state_s** structure is defined in **ve_sip_defs.h** and described below.

```
typedef struct
{
  t_ve_sip_dialog_field_state_e     field_state;
  t_ve_sip_dialog_state_e           dialog_state;
  t_ve_sip_dialog_state_event_e     event;
  t_ve_sip_response_code_e          code;
}t_ve_sip_dialog_state_s;
```

The elements of this structure are as follows:

Element	Definition
field_state	An enumerated type, **t_ve_sip_dialog_field_state_e** as described in Appendix B.199 on Page 619, that determines whether the destination has provided information about the state of the dialog.
dialog_state	An enumerated type, **t_ve_sip_dialog_state_e** as described in Appendix B.200 on Page 620, that identifies the state of the dialog.
event	An enumerated type, **t_ve_sip_dialog_state_event_e** as described in Appendix B.201 on Page 621, that optionally identifies the event that caused the change of state. This is normally provided when dialog_state is VE_SIP_DIALOG_STATE_TERMINATED.
code	An enumerated type, **t_ve_sip_response_code_e** as defined in Appendix B.181 on Page 602, that optionally identifies the SIP response that caused the change of state. This is normally provided when dialog_state is anything other than VE_SIP_DIALOG_STATE_TERMINATED.

B.48.1.3 t_ve_sip_dialog_direction_s

The **t_ve_sip_dialog_ direction_s** structure is part of the **t_ve_sip_dialog_s** structure described in Appendix B.48.1 on Page 460. The structure contains information about the direction of a SIP dialog. The **t_ve_sip_dialog_ direction_s** structure is defined in **ve_sip_defs.h** and described below.

```
typedef struct
{
  t_ve_sip_dialog_field_state_e     field_state;
  t_ve_sip_dialog_direction_e       direction;
}t_ve_sip_dialog_direction_s;
```

The elements of this structure are as follows:

Element	Definition
field_state	An enumerated type, **t_ve_sip_dialog_field_state_e** as described in Appendix B.199 on Page 619, that determines whether the destination has provided information about the direction of the dialog.
direction	An enumerated type, **t_ve_sip_dialog_direction_e** as described in Appendix B.202 on Page 622, that identifies the direction of the dialog.

B.48.1.4 t_ve_sip_dialog_duration_s

The **t_ve_sip_dialog_duration_s** structure is part of the **t_ve_sip_dialog_s** structure described in Appendix B.48.1 on Page 460. The structure contains information about the duration of a SIP dialog. The **t_ve_sip_dialog_duration_s** structure is defined in **ve_sip_defs.h** and described below.

```
typedef struct
{
  t_ve_sip_dialog_field_state_e   field_state;
  UWord32                         duration;
}t_ve_sip_dialog_duration_s;
```

The elements of this structure are as follows:

Element	Definition
field_state	An enumerated type, **t_ve_sip_dialog_field_state_e** as described in Appendix B.199 on Page 619, that determines whether the destination has provided information about the duration of the dialog.
duration	The duration of the dialog in seconds.

B.49 t_ve_sip_session_handle_s

The **t_ve_sip_session_handle_s** structure is used by the VeriCall Edge system SIP session functions, described in **Chapter 7**, to identify the specific SIP session. It is also contained in a number of the SIP specific user message events returned to the user application. The **t_ve_sip_session_handle_s** structure is defined in **ve_sip_defs.h** and described below:

```
typedef struct{
  UWord32    index;
}t_ve_sip_session_handle_s;
```

The element of this structure is as follows:

Element	Definition
index	An identification number supplied to the user application from the SIP stack to uniquely identify the SIP session.

B.50 t_ve_sip_privacy_identity_s

The **t_ve_sip_privacy_identity_s** structure is part of structure **t_ve_sip_call_id_s** as described in Appendix B.33.32.1 on Page 390, and structure **t_ve_remote_user_uri_s** as described in Appendix B.51 on Page 466. The structure describes a user's SIP privacy settings. The **t_ve_sip_privacy_identity_s** structure is defined in **ve_sip_defs.h** and described below:

```
typedef struct {
  UWord32                 num_privacy_values;
  t_ve_sip_privacy_s      privacy[VE_MAX_NUM_PRIVACY_VALUES];
  UWord32                 num_p_preferred_identities;
  t_ve_sip_user_uri_s
                sip_p_preferred_identity_uri[VE_MAX_NUM_IDENTITIES];
  UWord32                 num_p_asserted_identities;
  t_ve_sip_user_uri_s
                sip_p_asserted_identity_uri[VE_MAX_NUM_IDENTITIES];
}t_ve_sip_privacy_identity_s;
```

The element of this structure is as follows:

Element	Definition
num_privacy_values	The number of entries in the privacy array.
privacy	Structure of type t_ve_sip_privacy_s, as described in Appendix B.50.1 on Page 465, identifying the privacy setting that allows filtering of a user's name and address (implemented at the proxy or user agent).
num_p_preferred_identities	Number of entries in the sip_p_preferred_identity_uri array.
sip_p_preferred_identity_uri	Structure of type **t_ve_sip_user_uri_s**, as described in Appendix B.39.1 on Page 440, that populates the P-Preferred-Identity header.
num_p_asserted_identities	Number of entries in the sip_p_asserted_identity_uri array.
sip_p_asserted_identity_uri	Structure of type **t_ve_sip_user_uri_s**, as described in Appendix B.39.1 on Page 440, that poupulates the P-Asserted-Identity header.

B.50.1 t_ve_sip_privacy_s

The **t_ve_sip_privacy_s** structure is part of structure **t_ve_sip_privacy_identity_s** as described in Appendix B.50 on Page 464. The structure allows the application to control removal of P-Asserted-Identity header fields by the proxy server when transmitting to an element that is not trusted. The **t_ve_sip_privacy_s** structure is defined in **ve_sip_defs.h** and described below:

```
typedef struct {
  t_ve_sip_privacy_e        value;
}t_ve_sip_privacy_s;
```

The element of this structure is as follows:

Element	Definition
value	An enumerated type, **t_ve_sip_privacy_e** as described in Appendix B.204 on Page 624, that determines the SIP privacy requred.

B.51 t_ve_remote_user_uri_s

The **t_ve_remote_user_uri_s** structure is used by the VeriCall Edge system SIP session functions, described in **Chapter 7**, to define the remote SIP URI to be used for the specified SIP functionality. This structure is defined in **ve_sip_defs.h** and is described below:

```
typedef struct{
  Word8
     authentication_username[VE_MAX_AUTHENTICATION_USERNAME_SIZE];
  Word8   authentication_realm[VE_MAX_AUTHENTICATION_REALM_SIZE];
  Word8   password[VE_MAX_PASSWORD_SIZE];
  Word8   request_uri[VE_MAX_URI_STRING_SIZE];
  Word8   uri[VE_MAX_URI_STRING_SIZE];
  t_ve_sip_privacy_identity_s  privacy_identity;
  Word8   subject[VE_SIP_MAX_SUBJECT_STRING_SIZE];
  t_ve_sip_require_prack_e     require_prack;
  Word8   alert_info[VE_SIP_MAX_ALERT_INFO_STRING_SIZE];
}t_ve_remote_user_uri_s;
```

The elements of this structure are as follows:

Element	Definition
authentication_username	The user name user for authentication. This is only needed if it differs from the username in the SIP URI. If it is set to null then the user name from URI is used.
authentication_realm	Reserved for future use.
password	Optional authentication password, which can be NULL.
request_uri	Sets the request URI header. This allows the request URI to differ from the destinations URI in the *To:* header. If this is set to NULL then the request-URI will be the same as the URI. This field is useful for call forwarding.
uri	The remote URI which sets the *To:* header in SIP messages.
privacy_identity	Privacy and identity sets the Privacy:, P_Preferred_Identity and P_Asserted_Identity headers in SIP messages.
subject	Populates the subject SIP header. If this is NULL, no subject SIP header is included.
require_prack	An enumeration type, **t_ve_sip_require_prack_e** as described in Appendix B.185 on Page 606, which is a flag to inform the destination that PRACK is required to be used. This flag will only effect provisional responses. Note that the required header will only be added if the destination supports the feature.
alert_info	Populates the Alert-Info SIP header. If this is NULL then no Alert-Info is included. Alert-Info will only be added to an INVITE request.

B.52 t_ve_h323_user_registration_s

The **t_ve_h323_user_registration_s** structure is used in calls made to the **VE_H323AddUsersRegistrationData**, **VE_H323UpdateUsersRegistrationData** and **VE_H323GetUsersRegistrationData** functions. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. The structure allows for registration details of a user to be added to a H.323 registration server. The **t_ve_h323_user_registration_s** structure is defined in **ve_h323_defs.h** and is described below:

```
typedef struct{
  t_ve_h323_name_id_s            h323_name_id
  t_ve_h323_user_address_s       h323_user_address;
  t_ve_h323_contact_info_s       h323_contact_info;
}t_ve_h323_user_registration_s;
```

The elements of this structure are as follows:

Element	Definition
h323_name_id	A structure, **t_ve_h323_name_id_s** as defined in Appendix B.57 on Page 474, which defines the name identification information of this user.
h323_user_address	A structure, **t_ve_h323_user_address_s** as defined in Appendix B.52.1 on Page 467, which defines the name and IP address of this user.
h323_contact_info	A structure, **t_ve_h323_contact_info_s** as defined in Appendix B.52.2 on Page 468, which defines the optional contact information of this user.

B.52.1 t_ve_h323_user_address_s

The **t_ve_h323_user_address_s** structure is used within the **t_ve_h323_user_registration_s** structure as described in Appendix B.52 on Page 467. This structure allows a H.323 user address to be defined. The **t_ve_h323_user_address_s** structure is defined in **ve_h323_defs.h** and is described below:

```
typedef struct{
  Word8   address[VE_H323_MAX_ADDRESS_LEN];
}t_ve_h323_user_address_s;
```

The elements of this structure are as follows:

Element	Definition
address	A character array defining a users H.323 address, for example the user telephone number.

B.52.2 t_ve_h323_contact_info_s

The **t_ve_h323_contact_info_s** structure is used within the **t_ve_h323_user_registration_s** structure as described in Appendix B.52 on Page 467. This structure allows optional user contact information to be defined within the user application. The **t_ve_h323_contact_info_s** structure is defined in **ve_h323_defs.h** and is described below:

```
typedef struct{
 UWord16 num_contacts;
 Word8 contacts[VE_H323_MAX_ADDRESS_LEN][VE_H323_MAX_NUM_CONTACTS];
}t_ve_h323_contact_info_s;
```

The elements of this structure are as follows:

Element	Definition
num_contacts	The number of contacts associated with this user.
contacts	Multi-dimensional array holding the contact identifier and contact details of the user.

B.53 t_ve_h323_msg_center_id_s

The **t_ve_h323_msg_center_id_s** structure is used within the **t_ve_user_msg_h323_message_waiting_s** structure, as defined in Appendix B.33.48 on Page 402. The **t_ve_h323_msg_center_id_s** structure provides information and contact details of the Message Center. This structure is defined in **ve_h323_defs.h** and is described below:

```
typedef struct{
  Word32  number;
  Word8   party_number[VE_H323_MAX_MSG_CENTER_ID_PARTY_NUMBER_LEN];
  Word8
     numeric_string[VE_H323_MAX_MSG_CENTER_ID_NUMERIC_STRING_LEN];
  Word8   choice[VE_H323_MAX_MSG_CENTER_ID_CHOICE_LEN];
}t_ve_h323_msg_center_id_s;
```

The element of this structure is as follows:

Element	Definition
number	The format and value of this element is dependent on the Message Center application.
party_number	The format and value of this element is dependent on the Message Center application.
numeric_string	The format and value of this element is dependent on the Message Center application.
choice	The format and value of this element is dependent on the Message Center application.

The exact format of Message Center specific elements of this structure are dependent on the Message Center application. Please refer to the Message Center documentation for this information.

B.54 t_ve_h323_dest_address_s

The **t_ve_h323_dest_address_s** structure is used within a number of H.323 API calls, such as **VE_H323StartCall** as well as being contained within the **t_ve_user_msg_h323_message_waiting_s** structure, as defined in Appendix B.33.48 on Page 402. The **t_ve_h323_address_s** structure defines the destination H.323 address to be used within these API function calls. This structure is defined in **ve_h323_defs.h** and is described below:

```
typedef struct{
  Word8   address[VE_H323_MAX_ADDRESS_LEN];
}t_ve_h323_dest_address_s;
```

The element of this structure is as follows:

Element	Definition
address	An IPv4 or FQDN destination address.

The VeriCall Edge system uses the network byte ordering for IP addresses when a UWord32 value is required.

B.55 t_ve_h323_call_handle_s

The **t_ve_h323_call_handle_s** structure is used in H.323 API function calls to identify specific calls. The handle is a unique identifier assigned by the VeriCall Edge system, through the **VE_H323StartCall** function, and must be retained by the user application for the duration of the call (to allow use of other H.323 call related functions). The **t_ve_h323_call_handle_s** structure is defined in **ve_h323_defs.h** and is described below:

```
typedef struct{
  Word32     handle;
}t_ve_h323_call_handle_s;
```

The element of this structure is as follows:

Element	Definition
handle	A system provided unique H.323 call identifier.

B.56 t_ve_h323_call_update_user_channel_details_s

The **t_ve_h323_call_update_user_channel_details_s** structure is used in calls to the **VE_H323UpdateCall** function. The structure provides the users channel details that the channel is to be updated with. The **t_ve_h323_call_update_user_channel_details_s** structure is defined in **ve_h323_defs.h** and is described below:

```
typedef struct{
  UWord32       number_of_channels;
  t_ve_h323_call_update_user_channel_association_s
                channel_association[VE_MAX_CHANNEL_ASSOCIATION];
  UWord32       max_call_bandwidth_kbits_per_sec;
}t_ve_h323_call_update_user_channel_details_s;
```

The element of this structure is as follows:

Element	Definition
number_of_channels	The number of channels associated with this user. This must be the same as originally defined in the **t_ve_user_channel_details_s** structure, described in Appendix B.26 on Page 300.
channel_association	Structure of type **t_ve_h323_call_update_user_channel_associations_s** to define the new per channel algorithm preference associated with this user. The structure is defined in Appendix B.56.1 on Page 471. The number of channels that can be associated with this user cannot exceed VE_MAX_CHANNEL_ASSOCIATION that is defined as being 2.
max_call_bandwidth_kbits_per_sec	This element allows the user to update or define the maximum bandwidth available, shared over the number of channels associated with this user. The value is measured in kbps (k = 1000).

B.56.1 t_ve_h323_call_update_user_channel_associations_s

The **t_ve_h323_call_update_user_channel_associations_s** structure is used within the **t_ve_h323_call_update_user_channel_details_s** structure as described in Appendix B.56 on Page 470. This structure allows the update of a single channel associated with the user. The **t_ve_h323_call_update_user_channel_associations_s** structure is defined in **ve_h323_defs.h** and is described below:

```
typedef struct{
  t_ve_h323_user_channel_selection_s   channel_selection;
  t_ve_h323_call_update_attributes_s
            call_update_attributes[VE_MAX_NUM_ALG_PREF_LIST];
  UWord32                              num_call_update_attributes;
}t_ve_h323_call_update_user_channel_association_s;
```

The element of this structure is as follows:

Element	Definition
channel_selection	A structure of type **t_ve_h323_user_channel_selection_s** as described in Appendix B.56.2 on Page 472. This allows the channel and virtual sub-channel to be defined.
call_update_attributes	A structure of type **t_ve_h323_call_update_attributes_s** as described in Appendix B.56.3 on Page 473. This element allows the channels attributes to be updated.
num_call_update_attributes	The number of attributes within this channel to be updated. This is used to define the size of the array of the *call_update_attributes* element of this structure.

B.56.2 t_ve_h323_user_channel_selection_s

The **t_ve_h323_user_channel_selection_s** structure is used within the **t_ve_h323_call_update_user_channel_associations_s** structure as described in Appendix B.56.1 on Page 471. The structure allows the channel and sub-channel to be defined that are to be associated with the H.323 user. The **t_ve_h323_user_channel_selection_s** structure is defined in **ve_h323_defs.h** and is described below:

```
typedef struct{
  UWord32                                channel_number;
  t_ve_h323_user_sub_channel_select_e    sub_channel_select;
}t_ve_h323_user_channel_selection_s;
```

The element of this structure is as follows:

Element	Definition
channel_number	Defines the channel number against which the H.323 user is to be associated.
sub_channel_select	An enumerated media type, **t_ve_h323_user_sub_channel_select_e** as described in Appendix B.222 on Page 648, which allows the definition of the sub-channel to be associated with the H.323 user.

B.56.3 t_ve_h323_call_update_attributes_s

The **t_ve_h323_call_update_attributes_s** structure is used within the **t_ve_h323_call_update_user_channel_associations_s** structure as described in Appendix B.56.1 on Page 471. The structure allows channel attributes to be updated. The **t_ve_h323_call_update_attributes_s** structure is defined in **ve_h323_defs.h** and is described below:

```
typedef struct{
  t_ve_media_id_e                   media_id;
  t_ve_media_rtp_configuration_u    rtp_config;
}t_ve_h323_call_update_attributes_s;

typedef union{
  t_ve_media_g711_configuration_s      g711;
  t_ve_media_g729a_configuration_s     g729a;
  t_ve_media_g723_1_configuration_s    g723;
  t_ve_media_g726_configuration_s      g726;
  t_ve_media_gsm_amr_configuration_s   gsm_amr;
  t_ve_media_h263_configuration_s      h263;
  t_ve_media_mpeg4_configuration_s     mpeg4;
}t_ve_media_rtp_configuration_u;
```

The element of this structure is as follows:

Element	Definition
media_id	An enumerated media type, **t_ve_media_id_e** as defined in Appendix B.110 on Page 539, which is to be updated. The media type must be one that was originally added to this users channel via the **t_ve_channel_packet_configuration_s** structure described in Appendix B.21.3 on Page 293.
rtp_config	Union of structures defining the RTP configuration. This union is defined in **ve_defs.h** and the encapsulated structures of this union are as follows: **t_ve_media_g711_configuration_s** as defined in Appendix B.28 on Page 307. **t_ve_media_g729a_configuration_s** as defined in Appendix B.28.2 on Page 308. **t_ve_media_g723_1_configuration_s** as defined in Appendix B.28.3 on Page 309. **t_ve_media_g726_configuration_ s** as defined in Appendix 0 on Page 309. **t_ve_media_gsm_amr_configuration_s** as defined in Appendix B.28.3 on Page 309. **t_ve_media_h263_configuration_s** as defined in Appendix B.28.6 on Page 312. **t_ve_media_mpeg4_configuration_s** as defined in Appendix B.28.8 on Page 314.

B.57 t_ve_h323_name_id_s

The **t_ve_h323_name_id_s** structure is used within a number of **t_ve_user_msg_h323_xx_s** structures, such as the **t_ve_user_msg_h323_call_started_s** structure defined in Appendix B.33.37 on Page 395. This structure contains the H.450.8 name identification information. The **t_ve_h323_name_id_s** structure is defined in **ve_h323_defs.h** and is described below:

```
typedef struct{
  Word8                                 name[VE_H323_MAX_NAME_ID_LEN];
  t_ve_h323_name_id_char_set_e          char_set;
  t_ve_h323_name_id_restriction_e       restriction;
}t_ve_h323_name_id_s;
```

The elements of this structure are as follows:

Element	Definition
name	The name identification entry for this user, this can for example be the users telephone number.
char_set	A structure, **t_ve_h323_name_id_char_set_e** as defined in Appendix B.214 on Page 640 that defines the encoding method of *name*.
restriction	An enumerated type, **t_ve_h323_name_id_restriction_e** as defined in Appendix B.215 on Page 641 that determines restrictions on *name*.

B.58 t_ve_h323_default_configuration_s

The **t_ve_h323_default_configuration_s** structure is used for the VeriCall Edge system configuration to define the H.323 default configuration parameters. Table C.1 on Page 659 shows the VeriCall Edge attribute associated with this structure. The **t_ve_h323_default_configuration_s** structure is defined in **ve_h323_configuration.h** and is described below:

```
typedef struct{
  t_ve_h323_system_configuration_s     system;
  t_ve_h323_ras_configuration_s        ras;
  t_ve_h323_q931_configuration_s       q931;
  t_ve_h323_h245_configuration_s       h245;
  t_ve_h323_h450_configuration_s       h450;
}t_ve_h323_default_configuration_s;
```

The elements of the **t_ve_h323_default_configuration_s** structure are as follows:

Element	Definition
system	Structure of type **t_ve_h323_system_configuration_s**, as defined in Appendix B.58.1 on Page 476 that provides configuration options for the H.323 stack.
ras	Structure of type **t_ve_h323_ras_configuration_s**, as defined in Appendix B.58.2 on Page 477, which provides configuration options for H.323 RAS support.
q931	Structure of type **t_ve_h323_q931_configuration_s**, as defined in Appendix B.58.3 on Page 479 that provides configuration options for Q.931 support.
h245	Structure of type **t_ve_h323_h245_configuration_s**, as defined in Appendix B.58.4 on Page 482 that provides configuration options for H.245 support.
h450	Structure of type **t_ve_h323_h450_configuration_s**, as defined in Appendix B.58.5 on Page 483 that provides configuration options for H.450 support.

B.58.1 t_ve_h323_system_configuration_s

The **t_ve_h323_system_configuration_s** structure is used within the **t_ve_h323_default_configuration_s** structure as described in Appendix B.58 on Page 475. This structure defines the H.323 system configuration parameters. The **t_ve_h323_system_configuration_s** structure is defined in **ve_h323_defs.h** and is described below:

```
typedef struct{
  UWord16       maxbuffsize;
  UWord32       maxcalls;
  UWord32       maxpoolsize_in_kb;
  UWord32       maxChannels;
  UWord32       max_call_bandwidth_kbits_per_sec;
  UWord16       udp_start_port;
  UWord16       tcp_start_port;
  Word8         logging_conf[VE_H323_MAX_PATH_LEN];
  t_ve_h323_call_start_method_e     call_start_method;
}t_ve_h323_system_configuration_s;
```

The elements of the **t_ve_h323_system_configuration_s** structure are as follows:

Element	Definition
maxbuffsize	The maximum buffer size required for TPKT channels and RAS messages. This parameter indicates the maximum size of a message that can be sent or received by the H.323 stack. The default value is 2048.
maxcalls	The maximum number of concurrent H.323 calls allowed.
maxpoolsize_in_kb	The maximum size of encoded message buffers sent by the H.323 stack that are stored due to network congestion, measured in kilo-bytes per second. These messages can be resent when network resources become available. The default value is (*maxcalls* * *maxbuffsize*) / 1024.
maxChannels	The maximum number of concurrent channels allowed per H.323 call. Note that the channels are regarded as a pool of channels. This means that, for example, setting *maxcalls* to 10 and *maxchannels* to 2 will effectively create a pool of 20 channels for the use of all calls and not just two channels for each call.
max_call_bandwidth_kbits_per_sec	The absolute maximum bandwidth that the VeriCall Edge system will use, in 1000 * bits-per-second.
udp_start_port	The starting port number for UDP messages.
tcp_start_port	The starting port number for TCP messages.
logging_conf	The path to the configuration file for logging. If NULL no H.323 specific logging will take place. A valid path name will result in H.323 logging.
call_start_method	An enumerated type, **t_ve_h323_call_start_method_e** as described in Appendix B.220 on Page 646, which determines the H.323 call start method. This flag effects how an outgoing call is initiated.

B.58.2 t_ve_h323_ras_configuration_s

The **t_ve_h323_ras_configuration_s** structure is used within the **t_ve_h323_default_configuration_s** structure as described in Appendix B.58 on Page 475. This structure defines the Registration, Admission and Status protocol (H.225.0-RAS) configuration parameters. The **t_ve_h323_ras_configuration_s** structure is defined in **ve_h323_configuration.h** and is described below:

```
typedef struct{
  UWord16    response_time_out;
  Word16     t35CountryCode;
  Word16     t35Extension;
  Word16     manufacturerCode;
  Word8      productId[VE_H323_MAX_PRODUCT_ID_LEN];
  Word8      versionId[VE_H323_MAX_VERSION_ID_LEN];
  UWord32    time_to_live;
  t_ve_default_configuration_boolean_e   compare15bitRasCrv;
  UWord16    maxRetries;
  UWord16    maxMulticastTTL;
  UWord16    preTimeToLiveRegistration;
  t_ve_h323_gatekeeper_discovery_e  gatekeeper_discovery;
  t_ve_h323_gatekeeper_s            gatekeeper_primary;
  t_ve_h323_gatekeeper_s            gatekeeper_secondary;
}t_ve_h323_ras_configuration_s;
```

If the VeriCall Edge system is to be used without a H.323 gatekeeper, the `gatekeeper_discovery` field of this structure should be set to manual while the `gatekeeper_primary` and `gatekeeper_secondary` fields should be zeroed. The VE_H323RegisterUser function should not be called

Continued >>

The elements of the **t_ve_h323_ras_configuration_s** structure are as follows:

Element	Definition
response_time_out	The amount of time in seconds that the H.323 stack waits until it notifies the application that the called party has failed to respond to a transaction. The default value is 4 seconds.
t35CountryCode	Country code.
t35Extension	Extension.
manufacturerCode	The recognized manufacturer code.
productId	The product identifier.
versionId	The version identifier.
time_to_live	Determines the number of seconds before the registration becomes invalid. The endpoint will request from this value from the gatekeeper in RRQ messages.
compare15bitRasCrv	Causes a data-end point to ignore the CRV MSBit (the flag bit) in RAS messages. The default value is 16-bits.
maxRetries	The maximum number of retransmission retries. The default value is 3.
maxMulticastTTL	The maximum multicast time to live (TTL). The default value is 20 seconds.
preTimeToLiveRegistration	Sets the amount of time, in seconds, before the next TTL expires when the endpoint will attempt to re-register to the gatekeeper. The default is 1 second.
gatekeeper_discovery	Enumerated flag type, **t_ve_h323_gatekeeper_discovery_e** as defined in Appendix B.218 on Page 644, to indicate whether to use automatic or manual gatekeeper discovery.
gatekeeper_primary	Primary gatekeeper server details. This is a structure of type **t_ve_h323_gatekeeper_s**, as defined in Appendix B.58.2.1 on Page 478.
gatekeeper_secondary	Secondary (alternative) gatekeeper server details. This is a structure of type **t_ve_h323_gatekeeper_s**, as defined in Appendix B.58.2.1 on Page 478.

B.58.2.1 t_ve_h323_gatekeeper_s

The **t_ve_h323_gatekeeper_s** structure is used within the **t_ve_h323_ras_configuration_s** structure as described in Appendix B.58.2 on Page 477. The **t_ve_h323_gatekeeper_s** structure defines the H.323 gatekeeper server against which the users details are to be added. This structure is defined in **ve_h323_configuration.h** and is described below:

```
typedef struct{
  Word8      password[VE_H323_MAX_PASSWORD_SIZE];
  Word8      address[VE_H323_MAX_ADDRESS_LEN];
}t_ve_h323_gatekeeper_s;
```

The elements of this structure are as follows:

Element	Definition
password	Optional password, which can be NULL.
address	The IPv4 or FQDN address of the H.323 gatekeeper server.

The VeriCall Edge system uses the network byte ordering for IP addresses when a UWord32 value is required.

B.58.3 t_ve_h323_q931_configuration_s

The **t_ve_h323_q931_configuration_s** structure is used within the **t_ve_h323_default_configuration_s** structure as described in Appendix B.58 on Page 475. This structure defines the H.225.0-Q.931 configuration parameters. The **t_ve_h323_q931_configuration_s** structure is defined in **ve_h323_configuration.h** and is described below:

```
typedef struct{
  UWord16                                  responseTimeOut;
  UWord16                                  connectTimeOut;
  UWord16                                  callSignalingPort;
  UWord16                                  annexEcallSignalingPort;
  t_ve_default_configuration_boolean_e     h245Tunneling;
  t_ve_default_configuration_boolean_e     earlyH245;
  UWord16                                  t302;
  UWord16                                  t304;
  UWord16                                  t310;
  UWord16                                  t322;
  t_ve_h323_q931_annex_e_configuration_s   annex_e;
  UWord16                          postControlDisconnectionDelay;
  t_ve_default_configuration_boolean_e     keepCallOnQ931Close;
}t_ve_h323_q931_configuration_s;
```

Continued >>

The elements of this structure are as follows:

Element	Definition
responseTimeOut	The maximum waiting time for the first response to a new call. If no response is received during this time, the H.323 stack launches the disconnect procedure. The effect of a runtime change is immediate. Note: This is known as T304 in the Q.931 standard. The default value is 4 seconds.
connectTimeOut	The maximum time that the H.323 stack waits for the establishment of a new call, after receiving the first response to the call. If the call is not established during this time, the H.323 stack launches the disconnect procedure. The effect of a runtime change is immediate. Note: This is known as T301 in the Q.931 standard. The default value is 180 seconds.
callSignalingPort	The number of the port that receives the calls directed to the data-end point. If this parameter is set to 0, the H.323 stack looks for an available port. Using Q.931.multipleCallSignalingAddresses overrides this configuration parameter. The default value is 1720.
annexEcallSignalingPort	The Q.931 Annex E port to use. If this parameter is set to 0, the H.323 stack will choose any available port. The default value is 2517.
h245Tunneling	Enumerated type, **t_ve_default_configuration_boolean_e** as defined in Appendix B.84 on Page 521 that specifies whether a data-end point instance can support H.245 tunneling. If this parameter is set, tunneling is always to be used unless the other party does not support tunneling or the application specifically requests not to use tunneling. By default, this is disabled.
earlyH245	Enumerated type, **t_ve_default_configuration_boolean_e** as defined in Appendix B.84 on Page 521 that determines if Q.931 can send and accept an H.245 address in the setup message. By default, this parameter is disabled.
t302	The maximum waiting time for address data on the incoming side during an overlapped sending procedure. The default value is *responseTimeOut*, or 10 seconds if the *responseTimeOut* parameter is not set.
t304	The maximum waiting time for address data on the outgoing side during an overlapped sending procedure. The default value is *responseTimeOut*, or 4 seconds if the *responseTimeOut* parameter is not set.
t310	Timeout value between an incoming CallProceeding message and an Alerting or Connect message. The timeout is given in seconds with the default value being the *connectTimeOut* configuration parameter.
t322	Timeout value between a StatusEnquiry message that is sent and the received Status message reply. The timeout is given in seconds and defaults to 4 seconds.
annex_e	Structure of type **t_ve_h323_q931_annex_e_configuration_s**, as defined in Appendix B.58.3.1 on Page 481 that provides options for Annex E support.
postControlDisconnectionDelay	The amount of time, in milliseconds, to wait after disconnecting the H.245 session until disconnecting the Q.931 session. When this value is set to 0, the call will not be closed when Control disconnects. The default value is 2000 milliseconds.
keepCallOnQ931Close	Enumerated type, **t_ve_default_configuration_boolean_e** as defined in Appendix B.84 on Page 521 that indicates that closure of Q.931 does not necessitate dropping the call.

B.58.3.1 t_ve_h323_q931_annex_e_configuration_s

The **t_ve_h323_q931_annex_e_configuration_s** structure is used within the **t_ve_h323_q931_configuration_s** structure as described in Appendix B.58.3 on Page 479. This structure defines the H.225.0-Q.931 Annex E configuration parameters. The **t_ve_h323_q931_annex_e_configuration_s** structure is defined in **ve_h323_configuration.h** and is described below:

```
typedef struct{
  t_ve_default_configuration_boolean_e         use_annex_e;
  t_ve_h323_q931_annex_e_protocol_preference_e protocol_preference;
  UWord16    t_R1;
  UWord16    t_R2;
  UWord16    n_R1;
  UWord16    t_IMA1;
  UWord16    n_IMA1;
  UWord16    t_DT;
}t_ve_h323_q931_annex_e_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
use_annex_e	Enumerated type, **t_ve_default_configuration_boolean_e** as defined in Appendix B.84 on Page 521, that determines whether to use Annex E.
protocol_preference	Enumerated type, **t_ve_h323_q931_annex_e_protocol_preference_e** as defined in Appendix B.217 on Page 643 that determines which communication protocol to use for creating a call.
t_R1	The timeout for first re-send of a message (in milliseconds). The default is 500 milliseconds.
t_R2	The timeout for second re-send of a message (in milliseconds). The default is *t_R1* * 21/10.
n_R1	The maximum number of retransmissions before abandoning the connection. The default is 8.
t_IMA1	The timeout for retransmission of I-AM-ALIVE messages (in milliseconds). The default is 6000 milliseconds.
n_IMA1	The retry count for I-AM-ALIVE messages. The default is 6.
t_DT	The timeout, in milliseconds, to force sending messages while waiting for more building multiplexed messages. The default is 1 millisecond.

B.58.4 t_ve_h323_h245_configuration_s

The **t_ve_h323_h245_configuration_s** structure is used within the **t_ve_h323_default_configuration_s** structure as described in Appendix B.58 on Page 475. This structure defines the H.245 configuration parameters. The **t_ve_h323_h245_configuration_s** structure is defined in **ve_h323_configuration.h** and is described below:

```
typedef struct{
  UWord16    channelsTimeout;
  UWord16    roundTripTimeout;
  UWord16    requestCloseTimeout;
  UWord16    requestModeTimeout;
  UWord16    mediaLoopTimeout;
  UWord16    MasterSlave_timeOut;
  UWord16    capabilities_timeOut;
}t_ve_h323_h245_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
channelsTimeout	The timeout in seconds of the channel establishment. The default value is 29 seconds.
roundTripTimeout	The timeout in seconds of the round trip procedure. The default value is 10 seconds.
requestCloseTimeout	The timeout in seconds of the request close procedure. The default value is 29 seconds.
requestModeTimeout	The timeout in seconds of the request mode procedure. The default value is 10 seconds.
mediaLoopTimeout	The timeout in seconds of the media loop procedure. The default value is 10 seconds.
MasterSlave_timeOut	The maximum time in seconds that the H.323 stack will wait for a response to the master/slave inquiry before it decides it cannot perform the master/slave determination. The default value is 9 seconds.
capabilities_timeOut	The maximum time in seconds that the H.323 stack will wait before it decides that the called party will not acknowledge receipt of the capabilities it sent. The default value is 9 seconds.

B.58.5 t_ve_h323_h450_configuration_s

The **t_ve_h323_h450_configuration_s** structure is used within the **t_ve_h323_default_configuration_s** structure as described in Appendix B.58 on Page 475. This structure defines the H.450 configuration parameters. The **t_ve_h323_h450_configuration_s** structure is defined in **ve_h323_configuration.h** and is described below:

```
typedef struct{
  t_ve_h323_h450_2_configuration_s     h450_2;
  t_ve_h323_h450_3_configuration_s     h450_3;
  t_ve_h323_h450_4_configuration_s     h450_4;
  t_ve_h323_h450_6_configuration_s     h450_6;
  t_ve_h323_h450_7_configuration_s     h450_7;
}t_ve_h323_h450_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
h450_2	Structure of type **t_ve_h323_h450_2_configuration_s**, as defined in Appendix B.58.5.1 on Page 484 that provides configuration options for H.450.2 call transfer support.
h450_3	Structure of type **t_ve_h323_h450_3_configuration_s**, as defined in Appendix B.58.5.2 on Page 485 that provides configuration options for H.450.3 call diversion support.
h450_4	Structure of type **t_ve_h323_h450_4_configuration_s**, as defined in Appendix B.58.5.3 on Page 486, that provides configuration options for H.450.4 call hold support.
h450_6	Structure of type **t_ve_h323_h450_6_configuration_s**, as defined in Appendix B.58.5.4 on Page 486, that provides configuration options for H.450.6 call waiting support.
h450_7	Structure of type **t_ve_h323_h450_7_configuration_s**, as defined in Appendix B.58.5.5 on Page 487, that provides configuration options for H.450.7 call waiting indication support.

B.58.5.1 t_ve_h323_h450_2_configuration_s

The **t_ve_h323_h450_2_configuration_s** structure is used within the **t_ve_h323_h450_configuration_s** structure as described in Appendix B.58.5 on Page 483. This structure defines the H.450.2 call transfer configuration parameters. The **t_ve_h323_h450_2_configuration_s** structure is defined in **ve_h323_configuration.h** and is described below:

```
typedef struct{
  UWord16    ctt1;
  UWord16    ctt2;
  UWord16    ctt3;
  UWord16    ctt4;
}t_ve_h323_h450_2_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
ctt1	Timer CT-T1 operates at the TRGSE during state CT-Await-Identify-Response. Its purpose is to protect against the absence of a response to the CTIidentify.request.
ctt2	Timer CT-T2 operates at the TRTSE during state CT-Await-Setup. Its purpose is to protect against failure of completion of the call transfer operation.
ctt3	Timer CT-T3 operates at the TRGSE during state CT-Await-Initiate-Response. Its purpose is to protect against failure to establish the new connection.
ctt4	Timer CT-T4 may optionally operate at the TRDSE during state CT-Await-Setup-Response. Its purpose is to protect against failure to establish the new connection.

B.58.5.2 t_ve_h323_h450_3_configuration_s

The **t_ve_h323_h450_3_configuration_s** structure is used within the **t_ve_h323_h450_configuration_s** structure as described in Appendix B.58.5 on Page 483. This structure defines the H.450.3 call diversion configuration parameters. The **t_ve_h323_h450_3_configuration_s** structure is defined in **ve_h323_configuration.h** and is described below:

```
typedef struct{
  UWord16    cdt1;
  UWord16    cdt2;
  UWord16    cdt3;
  UWord16    cdt4;
  UWord16    cdt5;
}t_ve_h323_h450_3_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
cdt1	This timer is started by the served data-end point when a callRerouting invoked APDU is sent to the originating data-end point. The timer is stopped on receipt of a return result, return error or rejected APDU of the callRerouting operation. The expiry of this timer is equivalent to the receipt of a rejected APDU. This element should have a value of not less than 10 seconds.
cdt2	This timer is started by the activating data-end point when an activateDiversionQ invoked APDU is sent to the served data-end point. The timer is stopped on receipt of a return result, return error or rejected APDU of the activateDiversionQ operation. The expiry of this timer is equivalent to the receipt of a rejected APDU. This element should not have a value of less than 30 seconds.
cdt3	This timer is started by the deactivating data-end point when a deactivateDiversionQ invoked APDU is sent to the served data-end point. The timer is stopped on receipt of a return result, return error or rejected APDU of the deactivateDiversionQ operation. The expiry of this timer is equivalent to the receipt of a rejected APDU. This element should not have a value of less than 15 seconds.
cdt4	This timer is started by the interrogating data-end point when an interrogateDiversionQ invoked APDU is sent to the served data-end point. The timer is stopped on receipt of a return result, return error or rejected APDU of the interrogateDiversionQ operation. The expiry of this timer is equivalent to the receipt of a rejected APDU. This element should not have a value of less than 15 seconds.
cdt4	This timer is started by the served data-end point when a checkRestriction invoked APDU is sent to the diverted-to data-end point. The timer is stopped on receipt of a return result, return error or rejected APDU of the checkRestriction operation. The expiry of this timer is equivalent to the receipt of a rejected APDU. This element should not have a value of less than 15 seconds.

B.58.5.3 t_ve_h323_h450_4_configuration_s

The **t_ve_h323_h450_4_configuration_s** structure is used within the **t_ve_h323_h450_configuration_s** structure as described in Appendix B.58.5 on Page 483. This structure defines the H.450.4 call hold configuration parameters. The **t_ve_h323_h450_4_configuration_s** structure is defined in **ve_h323_configuration.h** and is described below:

```
typedef struct{
  UWord16    cht1;
  UWord16    cht2;
}t_ve_h323_h450_4_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
cht1	Timer CHT1 operates at the HGSE during state HGSE-RE-Requested. Its purpose is to protect against the absence of a response to the remoteHold.Invoke APDU request.
cht2	Timer CHT2 operates at the HGSE during state HGSE-RE-Retrieve-Requested. Its purpose is to protect against the absence of a response to the remoteRetrieve.Invoke APDU request.

B.58.5.4 t_ve_h323_h450_6_configuration_s

The **t_ve_h323_h450_6_configuration_s** structure is used within the **t_ve_h323_h450_configuration_s** structure as described in Appendix B.58.5 on Page 483. This structure defines the H.450.6 call waiting configuration parameters. The **t_ve_h323_h450_6_configuration_s** structure is defined in **ve_h323_configuration.h** and is described below:

```
typedef struct{
  UWord16    tcw;
}t_ve_h323_h450_6_configuration_s;
```

The element of this structure is as follows:

Element	Definition
tcw	Timer TCW operates at the WGSE during the WGSE-Call-Waiting-Invoked state. Its purpose is to protect against the absence of clearing the wait state of WDSE. The Timer is stopped when a Connect message is sent or when the call is released. If Timer TCW expires, sending a Release Complete message to the WDSE with the ReleaseCompleteReason set to "destinationRejection" clears the call.

B.58.5.5 t_ve_h323_h450_7_configuration_s

The **t_ve_h323_h450_7_configuration_s** structure is used within the **t_ve_h323_h450_configuration_s** structure as described in Appendix B.58.5 on Page 483. This structure defines the H.450.7 call waiting indication configuration parameters. The **t_ve_h323_h450_7_configuration_s** structure is defined in **ve_h323_configuration.h** and is described below:

```
typedef struct{
  UWord16    mwit1;
  UWord16    mwit2;
}t_ve_h323_h450_7_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
mwit1	Timer MWIT1 operates at the MWIMC during the MWI-MC-Wait state. Its purpose is to protect against the absence of a response to the mwiActivate or mwiDeactivate Invoke APDU.
mwit2	Timer MWIT2 operates at the MWISER during state MWI-Ser-Wait. Its purpose is to protect against the absence of a response to the mwiInterrogate Invoke APDU request.

B.59 t_ve_tone_definition_s

The **t_ve_tone_definition_s** structure is used by the **VE_GenerateUserTone** function and is used to define the user specified tone to be generated. The **t_ve_tone_definition_s** structure is defined in **ve_tones.h** and is described below:

```
typedef struct{
  UWord32                 duration;
  UWord8                  subgroup_count;
  UWord8                  repeat_count;
  t_ve_tone_subgroup_s    subgroup[VE_TONE_MAX_SUBGROUP_COUNT];
}t_ve_tone_definition_s;
```

The elements of this structure are as follows:

Element	Definition
duration	The duration of the tone to be generated, in milliseconds.
subgroup_count	The number of subgroup tones to be defined. This count dictates the size of the array of structures pointed to by the `subgroup` element of this structure.
repeat_count	The number of times the entire tone is to be repeated. A value of –1 indicates the tone is to be repeated indefinitely.
subgroup	A pointer to an array of structures containing the subgroup tone definitions. The size of the array of structures is defined by the `subgroup_count` element. This structure is of type **t_ve_tone_subgroup_s** and is described in Appendix B.59.1 on Page 489.

B.59.1 t_ve_tone_subgroup_s

The **t_ve_tone_subgroup_s** structure is used within the **t_ve_tone_definition_s** structure described in Appendix B.59 on Page 488. The **t_ve_tone_subgroup_s** structure is used to define a user specified sub-tone. This structure is defined in **ve_tones.h** and is described below.

```
typedef struct{
  UWord16                 duration;
  UWord8                  repeat_count;
  UWord8                  carrier;
  t_ve_tone_subtone_s     subtone[VE_TONE_MAX_SUBTONE_COUNT];
}t_ve_tone_subgroup_s;
```

The elements of this structure are as follows:

Element	Definition
duration	The duration of the sub-tone to be generated, in milliseconds.
repeat_count	The number of times the sub-tone(s) are to be repeated.
carrier	The carrier frequency of this sub-tone, in Hz.
subtone	An array of structures containing the sub-tone definitions. This structure is of type **t_ve_tone_subtone_s** and is described in Appendix B.59.2 on Page 490. The number of available sub tones is set by the `VE_TONE_MAX_SUBTONE_COUNT` constant which is defined as being four. Unused subtone values must be set to zero.

B.59.2 t_ve_tone_subtone_s

The **t_ve_tone_subtone_s** structure is used within the **t_ve_tone_subgroup_s** structure as described in Appendix B.59.1 on Page 489. The **t_ve_tone_subtone_s** structure is used to define the user specified sub-tone elements. This structure is defined in **ve_tones.h** and is described below.

```
typedef struct{
  UWord16       start_time;
  UWord32       end_time;
  UWord16       on_duration;
  UWord16       off_duration;
  UWord16       frequency;
  UWord16       divisor;
  Word16        scale;
  UWord16       phase_period;
}t_ve_tone_subtone_s;
```

The elements of this structure are as follows:

Element	Definition
start_time	The start time of the sub-tone, in milliseconds.
end_time	The end time of the sub-tone, in milliseconds.
on_duration	The duration the sub-tone is to “on”, in milliseconds.
off_duration	The duration the sub-tone is to be “off”, in milliseconds.
frequency	The frequency of the sub-tone, in Hz.
divisor	Integer divisor of the sub-tone.
scale	Integer scale of the sub-tone.
phase_period	The phase-period of the subtone, in milliseconds.

B.60 t_ve_nat_default_configuration_s

The **t_ve_nat_default_configuration_s** structure is used for the VeriCall Edge system configuration to define the NAT traversal module configuration parameters. Table C.1 on Page 659 shows the VeriCall Edge attribute associated with this structure. This structure is defined in **ve_nat_configuration.h** and is described below:

```
typedef struct{
  UWord16       port_range_start;
  UWord16       port_range_end;
  UWord32       keep_alive_period;
  UWord32       keep_alive_optimise;
  UWord32       binding_lifetime_accuracy;
  UWord32       binding_lifetime_maximum;
}t_ve_nat_default_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
port_range_start	The start range port number of the range of local ports available for use by the NAT traversal module. At least one pair of ports must be made available.
port_range_end	The end range port number of the range of local ports available for use by the NAT traversal module. At least one pair of ports must be made available.
keep_alive_period	The default keep-alive period to use (seconds). This value defaults to 20 seconds. If this value is zero, this means no keep-alive mechanism will be employed by the NAT Traversal module.
keep_alive_optimise	The percentage of binding lifetime to automatically optimize the keep-alive period to when binding lifetime is discovered. If this value is zero, this means do not optimize.
binding_lifetime_accuracy	The accuracy to use when calculating the binding lifetime (seconds). If this value is zero, this means there is no binding lifetime discovery.
binding_lifetime_maximum	The maximum to use when calculating the binding lifetime (seconds). If this value is zero, this means there is no binding lifetime discovery.

B.61 t_ve_nat_channel_addresses_s

The **t_ve_nat_channel_addresses_s** structure is used by the **VE_NATSetChannelAddressesData** and **VE_NATGetChannelAddressesData** function as well as within the **t_ve_user_msg_nat_user_addresses_response_s** structure, described in Appendix B.33.2 on Page 360. This structure contains the public IP assigned addresses. The **t_ve_nat_channel_addresses_s** structure is defined in **ve_nat_defs.h** and is described below:

```
typedef struct{
  t_ve_nat_media_addresses_s        stun;
  t_ve_nat_media_addresses_s        turn;
  t_ve_nat_media_addresses_s        manual;
}t_ve_nat_channel_addresses_s;
```

The elements of this structure are as follows:

Element	Definition
stun	A structure, **t_ve_nat_media_addresses_s** as described in Appendix B.61.1 on Page 493, which defines the public IP STUN address mapping for the specified channel.
turn	A structure, **t_ve_nat_media_addresses_s** as described in Appendix B.61.1 on Page 493, which defines the public IP TURN address mapping for the specified channel. Reserved for future use.
manual	A structure, **t_ve_nat_media_addresses_s** as described in Appendix B.61.1 on Page 493, which defines a manual public IP address mapping for the specified channel. This means the user application can set the IP address to be used rather than using STUN or TURN. Use of this address will override all other addresses used.

B.61.1 t_ve_nat_media_addresses_s

The **t_ve_nat_media_addresses_s** structure is used within the **t_ve_nat_channel_addresses_s** structure described in Appendix B.61 on Page 492. This structure contains the IP address information. The **t_ve_nat_media_addresses_s** structure is defined in **ve_nat_defs.h** and is described below:

```
typedef struct{
  t_ve_nat_state_e                  state;
  t_ve_media_ip_configuration_s     rtp_ip;
  UWord16                           rtp_port;
  t_ve_media_ip_configuration_s     rtcp_ip;
  UWord16                           rtcp_port;
}t_ve_nat_media_addresses_s;
```

The elements of this structure are as follows:

Element	Definition
state	An enumerated value, of type **t_ve_nat_state_e** as described in Appendix B.228 on Page 651, used to determine if the specified public IP address mapping is valid. Note if this enumerated type is enabled, this means the mappings are valid.
rtp_ip	A structure, **t_ve_media_ip_configuration_s** as described in Appendix B.27.2 on Page 304, which defines the RTP mapped address to be used within this mapping.
rtp_port	The RTP port number to be used with this mapping.
rtcp_ip	A structure, **t_ve_media_ip_configuration_s** as described in Appendix B.27.2 on Page 304, which defines the RTCP mapped address to be used within this mapping.
rtcp_port	The RTCP port number to be used with this mapping.

B.62 t_ve_nat_init_s

The **t_ve_nat_init_s** structure is used as an argument to the **VE_NATStartDiscover** API function. This structure initializes the VeriCall Edge NAT traversal module. The **t_ve_nat_init_s** structure is defined in **ve_nat_defs.h** and is described below:

```
typedef struct{
  t_ve_nat_discover_method_e      discover_method;
  t_ve_nat_stun_init_s            stun;
  t_ve_nat_turn_init_s            turn;
}t_ve_nat_init_s;
```

The elements of this structure are as follows:

Element	Definition
discover_method	An enumerated value, of type **t_ve_nat_discover_method_e** as described in Appendix B.227 on Page 650, specifying the NAT discovery method to be used.
stun	A structure, **t_ve_nat_stun_init_s** as described in Appendix B.62.1 on Page 495, which initializes the STUN server connection parameters.
turn	A structure, **t_ve_nat_turn_init_s** as described in Appendix B.62.2 on Page 495, which initializes the TURN server connection parameters. Reserved for future use.

B.62.1 t_ve_nat_stun_init_s

The **t_ve_nat_stun_init_s** structure is used within the **t_ve_nat_init_s** structure described in Appendix B.62 on Page 494. This structure configure the STUN server connection parameters. The **t_ve_nat_stun_init_s** structure is defined in **ve_nat_defs.h** and is described below:

```
typedef struct{
  t_ve_nat_state_e                  state;
  t_ve_media_ip_configuration_s     ip;
  UWord16                           port;
}t_ve_nat_stun_init_s;
```

The elements of this structure are as follows:

Element	Definition
state	An enumerated value, of type **t_ve_nat_state_e** as described in Appendix B.228 on Page 651, specifying the state of the NAT STUN client. Note that if this is enabled this means that the STUN client details are available.
ip	A structure, **t_ve_media_ip_configuration_s** as described in Appendix B.27.2 on Page 304, which determines the IP address of the STUN server.
port	Allows the STUN port number to be defined. If this value is zero, the standard port number, 3478, will be used.

The standard STUN port number, 3478, is as defined in the RFC3489.

B.62.2 t_ve_nat_turn_init_s

The **t_ve_nat_turn_init_s** structure is used within the **t_ve_nat_init_s** structure described in Appendix B.62 on Page 494. This structure initializes the TURN server connection parameters. The **t_ve_nat_turn_init_s** structure is defined in **ve_nat_defs.h** and is described below:

```
typedef struct{
  t_ve_nat_state_e     state;
}t_ve_nat_turn_init_s;
```

The elements of this structure are as follows:

Element	Definition
state	An enumerated value, of type **t_ve_nat_state_e** as described in Appendix B.228 on Page 651, specifying the state of the NAT TURN client. Note that if this is enabled this means that the TURN client details are available. Reserved for future use.

B.63 t_ve_nat_configuration_s

The **t_ve_nat_configuration_s** structure is used as an argument to the **VE_NATSetDiscoverData** and **VE_NATGetDiscoverData** API functions, as well as within the **t_ve_user_msg_nat_discover_response_s** structure which is described in Appendix B.33.1 on Page 360. This structure contains configuration details of the NAT device behind which the VeriCall Edge system resides. The **t_ve_nat_configuration_s** structure is defined in **ve_nat_defs.h** and is described below:

```
typedef struct{
  t_ve_nat_types_e              nat_type;
  UWord32                       flags;
  UWord32                       binding_lifetime;
  t_ve_nat_state_e              stun_state;
  t_ve_nat_state_e              turn_state;
  t_ve_nat_sip_addresses_s      sip;
}t_ve_nat_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
nat_type	An enumerated value, of type **t_ve_nat_types_e** as described in Appendix B.229 on Page 652, specifying the type of NAT behind which the VeriCall Edge system resides.
flags	A bit mask containing extra information about the NAT. The bit mask values can be the following: VE_NAT_SUPPORTS_HAIRPINNING 0x01 VE_NAT_PRESERVES_PORT_NUMBERS 0x02
binding_lifetime	The binding lifetime of the NAT, in seconds.
stun_state	An enumerated value, of type **t_ve_nat_state_e** as described in Appendix B.228 on Page 651, specifying the state of the NAT traversal STUN client. Note that if this is enabled this means that the client is active.
turn_state	An enumerated value, of type **t_ve_nat_state_e** as described in Appendix B.228 on Page 651, specifying the state of the NAT traversal TURN client. Note that if this is enabled this means that the client is active. Reserved for future use.
sip	A structure, **t_ve_nat_sip_addresses_s** as described in Appendix B.63.1 on Page 497, which allows the SIP messaging public addresses to be overridden.

NAT hairpinning is the ability to turn-around publicly addressed packets, destined for a location *inside* the NAT, back to the correct internal destination.

NAT can preserve port numbers from the client session and use the same port number on the NAT public address if it is available.

B.63.1 t_ve_nat_sip_addresses_s

The **t_ve_nat_sip_addresses_s** structure is used within the **t_ve_nat_configuration_s** structure described in Appendix B.63 on Page 496. This structure contains the mapped SIP addresses used by the VeriCall Edge NAT traversal module. The **t_ve_nat_sip_addresses_s** structure is defined in **ve_nat_defs.h** and is described below:

```
typedef struct{
  t_ve_nat_state_e                  state;
  t_ve_media_ip_configuration_s     sig_ip;
  UWord16                           sig_port;
}t_ve_nat_sip_addresses_s;
```

The elements of this structure are as follows:

Element	Definition
state	An enumerated value, of type **t_ve_nat_state_e** as described in Appendix B.228 on Page 651, used to determine whether the SIP public IP address is valid. Note that if this is enabled this means that the SIP public IP address is valid.
sig_ip	A structure, **t_ve_media_ip_configuration_s** as described in Appendix B.27.2 on Page 304, which defines the SIP signaling mapped IP address.
sig_port	Defines the SIP signaling mapped port number.

B.64 t_ve_network_connection_s

The **t_ve_network_connection_s** structure is used by calls to the **VE_NATStartKeepAlive** and **VE_NATStopKeepAlive** functions. This function is also used within the **t_ve_user_msg_network_connection_event_s** structure, described in Appendix B.33.64 on Page 417, and the **t_ve_user_msg_nat_keep_alive_event_s** structure, described in Appendix B.33.4 on Page 362. This structure allows the configuration of the NAT network connection which is to be kept alive. This structure is defined in **ve_defs.h** and is described below:

```
typedef struct{
  t_ve_network_connection_type_e    type;
  t_ve_media_ip_configuration_s     local_ip;
  UWord16                           local_port;
  t_ve_media_ip_configuration_s     remote_ip;
  UWord16                           remote_port;
}t_ve_network_connection_s;
```

The elements of this structure are as follows:

Element	Definition
type	An enumerated value, of type **t_ve_network_connection_type_e** as described in Appendix B.135 on Page 561, used to determine the network connection type.
local_ip	A structure, **t_ve_media_ip_configuration_s** as described in Appendix B.27.2 on Page 304, which defines the media configuration of the local VeriCall Edge IP end-point.
local_port	The port number of the local VeriCall Edge IP end-point.
remote_ip	A structure, **t_ve_media_ip_configuration_s** as described in Appendix B.27.2 on Page 304, which defines the media IP configuration of the remote end-point.
remote_port	The port number of the remote IP end-point.

B.65 t_ve_net_default_configuration_s

The **t_ve_net_default_configuration_s** structure is used for the VeriCall Edge system configuration to define the default network services configuration for the VeriCall Edge system. Table C.1 on Page 659 shows the VeriCall Edge attribute association with this structure. This structure is defined in **ve_net_configuration.h** and is described below:

```
typedef struct{
  t_ve_media_ip_configuration_s     primary_dns_ip;
  t_ve_media_ip_configuration_s     secondary_dns_ip;
  UWord16                           dns_port;
  UWord32                           time_out_period;
  UWord32                           time_out_max_retries;
}t_ve_net_default_configuration_s;
```

The elements of this structure are as follows:

Element	Definition
primary_dns_ip	A structure, **t_ve_media_ip_configuration_s** as described in Appendix B.27.2 on Page 304, which defines the primary DNS server IP address to be used.
secondary_dns_ip	A structure, **t_ve_media_ip_configuration_s** as described in Appendix B.27.2 on Page 304, which defines the secondary DNS server IP address to be used.
dns_port	The local IP port available for use for DNS
time_out_period	The time-out period, defined in milliseconds.
time_out_max_retries	The maximum number of timeout retries to be performed.

At least a primary or secondary DNS IP address must be set by the user application before DNS operations can be invoked.

B.66 t_ve_net_lookup_query_s

The **t_ve_net_lookup_query_s** structure is used as an argument in calls to **VE_NETLookUp**, as well as within the **t_ve_user_msg_net_lookup_response_s** structure, described in Appendix B.33.65 on Page 418. This structure contains the information to be used for the network lookup. This structure is defined in **ve_net_defs.h** and is described below:

```
typedef struct{
  UWord32                                transaction_id;
  t_ve_net_lookup_type_e                 type;
  union{
     t_ve_net_lookup_ipv4_address_s      ipv4_address;
     t_ve_net_lookup_server_s            server;
     t_ve_net_lookup_naptr_s             naptr;
     t_ve_net_lookup_naptr_enum_s        naptr_enum;
  }data;
}t_ve_net_lookup_query_s;
```

The elements of this structure are as follows:

Element	Definition
transaction_id	The client transaction identifier.
type	An enumerated value, of type **t_ve_net_lookup_type_e** as described in Appendix B.137 on Page 563, used to determine the network lookup type.
data	A union of structures whose selection depends on the value of the *type* element. The selected structure can be one of the following: • Structure **t_ve_net_lookup_ipv4_address_s** as described in Appendix B.67 on Page 501. This structure defines the IPv4 address to be looked up. • Structure **t_ve_net_lookup_server_s** as described in Appendix B.68 on Page 501. This structure defines the service type, service protocol and domain to be used in the DNS SVR lookup. • Structure **t_ve_net_lookup_naptr_s** as described in Appendix B.69 on Page 502. This structure contains the NAPTR service type, service protocol and domain to be used in the NAPTR lookup. • Structure **t_ve_net_lookup_naptr_enum_s** as described in Appendix B.70 on Page 503. This structure contains the NAPTR service type, ENUM telephone number and domain to be used in the NAPTR ENUM lookup.

B.67 t_ve_net_lookup_ipv4_address_s

The **t_ve_net_lookup_ipv4_address_s** structure is used within the **t_ve_net_lookup_query_s** structure, described in Appendix B.66 on Page 500. This structure contains the Fully Qualified Domain Name (FQDN) to be used in this network lookup request. This structure is defined in **ve_net_defs.h** and is described below:

```
typedef struct{
  Word8         name[VE_MAX_URI_STRING_SIZE];
}t_ve_net_lookup_ipv4_address_s;
```

The element of this structure is as follows:

Element	Definition
name	The FQDN to be “looked-up” – translated to an IPv4 address.

The VeriCall Edge system uses the network byte ordering for IP addresses when a UWord32 value is required.

B.68 t_ve_net_lookup_server_s

The **t_ve_net_lookup_server_s** structure is used within the **t_ve_net_lookup_query_s** structure, described in Appendix B.66 on Page 500. This structure contains the server information to be used in this network lookup request. This structure is defined in **ve_net_defs.h** and is described below:

```
typedef struct{
  t_ve_net_service_type_e       service_type;
  t_ve_net_service_protocol_e   service_proto;
  Word8                         domain[VE_MAX_URI_STRING_SIZE];
}t_ve_net_lookup_server_s;
```

The elements of this structure are as follows:

Element	Definition
service_type	An enumerated value, of type **t_ve_net_service_type_e** as described in Appendix B.138 on Page 564, used to determine the network service type.
service_proto	An enumerated value, of type **t_ve_net_service_protocol_e** as described in Appendix B.139 on Page 565, used to determine the network service type.
domain	The network domain name in which the service resides. The name string must not exceed `VE_MAX_URI_STRING_SIZE`, including the termination character.

The VeriCall Edge system uses the network byte ordering for IP addresses when a UWord32 value is required.

B.69 t_ve_net_lookup_naptr_s

The **t_ve_net_lookup_naptr_s** structure is used within the **t_ve_net_lookup_query_s** structure, described in Appendix B.66 on Page 500. This structure contains the NAPTR information to be used in this network lookup request. This structure is defined in **ve_net_defs.h** and is described below:

```
typedef struct{
  t_ve_net_service_type_e      service_type;
  t_ve_net_service_protocol_e  service_proto;
  Word8                        domain[VE_MAX_URI_STRING_SIZE];
}t_ve_net_lookup_naptr_s;
```

The elements of this structure are as follows:

Element	Definition
service_type	An enumerated value, of type **t_ve_net_service_type_e** as described in Appendix B.138 on Page 564, used to determine the network service type.
service_proto	An enumerated value, of type **t_ve_net_service_protocol_e** as described in Appendix B.139 on Page 565, used to determine the network service type.
domain	The domain name within which the NAPTR service resides. The name string must not exceed `VE_MAX_URI_STRING_SIZE`, including the termination character.

The VeriCall Edge system uses the network byte ordering for IP addresses when a UWord32 value is required.

B.70 t_ve_net_lookup_naptr_enum_s

The **t_ve_net_lookup_naptr_enum_s** structure is used within the **t_ve_net_lookup_query_s** structure, described in Appendix B.66 on Page 500. This structure contains the NAPTR ENUM information to be used in this network lookup request. This structure is defined in **ve_net_defs.h** and is described below:

```
typedef struct{
  Word8                     phone_number[VE_MAX_E164_STRING_SIZE];
  t_ve_net_service_type_e   service_type;
  Word8                     domain[VE_MAX_URI_STRING_SIZE];
}t_ve_net_lookup_naptr_enum_s;
```

The elements of this structure are as follows:

Element	Definition
phone_number	The E.164 phone number to be used in this lookup request. The phone number string must not exceed `VE_MAX_E164_STRING_SIZE`, including the termination character.
service_type	An enumerated value, of type **t_ve_net_service_type_e** as described in Appendix B.138 on Page 564, used to determine the network service type.
domain	The domain name within which the NAPTR ENUM service resides. The name string must not exceed `VE_MAX_URI_STRING_SIZE`, including the termination character.

The VeriCall Edge system uses the network byte ordering for IP addresses when a UWord32 value is required.

VeriCall Edge Enumerations

B.71 t_ve_attribute_type_e

The **t_ve_attribute_type_e** enumeration is used in calls to the following API functions:

- **VE_SetSystemAttribute**
- **VE_GetSystemAttribute**
- **VE_UpdateSystemAttribute**
- **VE_UpdateChannelAttribute**
- **VE_RemoveSystemAttribute**
- **VE_UpdateUserAttribute**

The enumeration is used to identify specific attributes that configure the VeriCall Edge system. Associated structures to configure the VeriCall Edge system through these attributes are stated in Table C.1 on Page 659, while Table C.2 on Page 662 shows the associated index values used to access these attributes. The enumeration is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_ATTRIBUTE_TYPE_NO_VALUE = 0,
  VE_ATTRIBUTE_TYPE_INITIALIZATION_ATTRIBUTES,
  VE_ATTRIBUTE_TYPE_DEFAULT_IP,
  VE_ATTRIBUTE_TYPE_DEFAULT_IP_SETTINGS,
  VE_ATTRIBUTE_TYPE_DEFAULT_SYSTEM_IP_SETTINGS,
  VE_ATTRIBUTE_TYPE_DEFAULT_SIP_IP_SETTINGS,
  VE_ATTRIBUTE_TYPE_DEFAULT_H323_IP_SETTINGS,
  VE_ATTRIBUTE_TYPE_DEFAULT_NAT_IP_SETTINGS,
  VE_ATTRIBUTE_TYPE_DEFAULT_NET_IP_SETTINGS,
  VE_ATTRIBUTE_TYPE_DEFAULT_SIP_CONFIGURATION,
  VE_ATTRIBUTE_TYPE_MEDIA_PROFILE,
  VE_ATTRIBUTE_TYPE_DEFAULT_AJBM,
  VE_ATTRIBUTE_TYPE_DEFAULT_PRE_PROC_CONFIGURATION,
  VE_ATTRIBUTE_TYPE_DEFAULT_POST_PROC_CONFIGURATION,
  VE_ATTRIBUTE_TYPE_DEFAULT_NTG,
  VE_ATTRIBUTE_TYPE_DEFAULT_NTD,
  VE_ATTRIBUTE_TYPE_DEFAULT_LINE_ECAN,
  VE_ATTRIBUTE_TYPE_CHANNEL_CONFIGURATION,
  VE_ATTRIBUTE_TYPE_USER_TRANSPORT_OPEN_FUNCTION,
  VE_ATTRIBUTE_TYPE_USER_TRANSPORT_SEND_FUNCTION,
  VE_ATTRIBUTE_TYPE_USER_TRANSPORT_RECEIVE_FUNCTION,
  VE_ATTRIBUTE_TYPE_USER_TRANSPORT_CLOSE_FUNCTION,
  VE_ATTRIBUTE_TYPE_USER_TRANSPORT_CONFIGURE_FUNCTION,
```

Continued >>

```
    VE_ATTRIBUTE_TYPE_USER_DATA_CHANNEL_TRANSFER_OPEN_FUNCTION,
    VE_ATTRIBUTE_TYPE_USER_DATA_CHANNEL_DATA_TRANSFER_FUNCTION,
    VE_ATTRIBUTE_TYPE_USER_DATA_CHANNEL_TRANSFER_CLOSE_FUNCTION,
    VE_ATTRIBUTE_TYPE_USER_DATA_CHANNEL_TRANSFER_AVAIL_FUNCTION,
    VE_ATTRIBUTE_TYPE_USER_DATA_CHANNEL_TRANSFER_QUERY_FUNCTION,
    VE_ATTRIBUTE_TYPE_USER_DATA_CHANNEL_TRANSFER_CONFIG_FUNCTION,
    VE_ATTRIBUTE_TYPE_READONLY_COMMIT_LOCK,
    VE_ATTRIBUTE_TYPE_READONLY_INITIALIZE_LOCK,
    VE_ATTRIBUTE_TYPE_READONLY_USER,
    VE_ATTRIBUTE_TYPE_READONLY_STATISTICS_SYSTEM_CALLS,
    VE_ATTRIBUTE_TYPE_READONLY_STATISTICS_VIDEO_ENCODER,
    VE_ATTRIBUTE_TYPE_READONLY_STATISTICS_VIDEO_DECODER,
    VE_ATTRIBUTE_TYPE_READONLY_STATISTICS_RTCP,
    VE_ATTRIBUTE_TYPE_READONLY_STATISTICS_RTCP_PEER,
    VE_ATTRIBUTE_TYPE_READONLY_STATISTICS_AV_SYNC,
    VE_ATTRIBUTE_TYPE_READONLY_USER_SIP_DETAILS,
    VE_ATTRIBUTE_TYPE_READONLY_DEFAULT_STREAM_STRUCTURE,
    VE_ATTRIBUTE_TYPE_READONLY_LAST_START_STREAM_STRUCTURE,
    VE_ATTRIBUTE_TYPE_DEFAULT_DATA_END_POINT,
    VE_ATTRIBUTE_TYPE_DEFAULT_TX_LIP_SYNC,
    VE_ATTRIBUTE_TYPE_READONLY_USER_H323_DETAILS,
    VE_ATTRIBUTE_TYPE_DEFAULT_H323_CONFIGURATION,
    VE_ATTRIBUTE_TYPE_SYTEM_LOGGING,
    VE_ATTRIBUTE_TYPE_DEFAULT_ACOUSTIC_ECAN,
    VE_ATTRIBUTE_TYPE_DEFAULT_VIDEO_TEAR_PROTECT,
    VE_ATTRIBUTE_TYPE_DEFAULT_NAT_CONFIGURATION,
    VE_ATTRIBUTE_TYPE_READONLY_USER_SIP_PRESENCE_DETAILS,
    VE_ATTRIBUTE_TYPE_READONLY_USER_SIP_MW_DETAILS,
    VE_ATTRIBUTE_TYPE_DEFAULT_NET_CONFIGURATION,
    VE_ATTRIBUTE_TYPE_DEFAULT_GAIN_CONTROL_TX,
    VE_ATTRIBUTE_TYPE_DEFAULT_GAIN_CONTROL_RX,
    VE_ATTRIBUTE_TYPE_NUMBER_OF_ATTRIBUTES
}t_ve_attribute_type_e;
```

Read-only attributes can only be accessed by calls to VE_GetSystemAttribute.

Continued >>

The enumeration elements are as follows:

Attribute	Structure Association
VE_ATTRIBUTE_TYPE_ NO_VALUE	Not accessible.
VE_ATTRIBUTE_TYPE_ INITIALIZATION_ATTRIBUTES	The VeriCall Edge system initialization parameters encapsulated in the **t_ve_init_config_s** structure, described in Appendix B.2 on Page 254.
VE_ATTRIBUTE_TYPE_ DEFAULT_IP	The default IP address of the VeriCall Edge system encapsulated in the **t_ve_ip_default_configuration_s** structure, described in Appendix B.7 on Page 260
VE_ATTRIBUTE_TYPE_ DEFAULT_IP_SETTINGS	The default IP settings for the VeriCall Edge system encapsulated in the **t_ve_ip_settings_default_configuration_s** structure, described in Appendix B.8 on Page 261.
VE_ATTRIBUTE_TYPE_ DEFAULT_SYSTEM_IP_SETTINGS	Optional IP settings for the VeriCall Edge Media Engine encapsulated in the **t_ve_ip_settings_default_configuration_s** structure, described in Appendix B.8 on Page 261.
VE_ATTRIBUTE_TYPE_ DEFAULT_SIP_IP_SETTINGS	Optional IP settings for the VeriCall Edge SIP module encapsulated in the **t_ve_ip_settings_default_configuration_s** structure, described in Appendix B.8 on Page 261.
VE_ATTRIBUTE_TYPE_ DEFAULT_H323_IP_SETTINGS	Optional IP settings for the VeriCall Edge H.323 module encapsulated in the **t_ve_ip_settings_default_configuration_s** structure, described in Appendix B.8 on Page 261.
VE_ATTRIBUTE_TYPE_ DEFAULT_NAT_IP_SETTINGS	Optional IP settings for the VeriCall Edge NAT traversal module encapsulated in the **t_ve_ip_settings_default_configuration_s** structure, described in Appendix B.8 on Page 261.
VE_ATTRIBUTE_TYPE_ DEFAULT_NET_IP_SETTINGS	Optional IP settings for the VeriCall Edge Network Services module encapsulated in the **t_ve_ip_settings_default_configuration_s** structure, described in Appendix B.8 on Page 261.
VE_ATTRIBUTE_TYPE_ DEFAULT_SIP_CONFIGURATION	The SIP configuration settings of the VeriCall Edge system encapsulated in the **t_ve_sip_default_configuration_s** defined in Appendix B.17 on Page 284.
VE_ATTRIBUTE_TYPE_ MEDIA_PROFILE	The media configurations defined in the VeriCall Edge system encapsulated in the **t_ve_media_definition_s** defined in Appendix B.29 on Page 315.
VE_ATTRIBUTE_TYPE_ DEFAULT_AJBM	The AJBM configuration encapsulated in the **t_ve_ajbm_algorithm_default_configuration_s** defined in Appendix B.9 on Page 262.
VE_ATTRIBUTE_TYPE_ DEFAULT_PRE_PROC_CONFIGURATION	The VeriCall Edge system pre-processor configuration encapsulated in the **t_ve_pre_proc_default_configuration_s** defined in Appendix B.15 on Page 275.
VE_ATTRIBUTE_TYPE_ DEFAULT_POST_PROC_CONFIGURATION	The VeriCall Edge system post-processor configuration encapsulated in the **t_ve_post_proc_default_configuration_s** defined in Appendix B.16 on Page 280.
VE_ATTRIBUTE_TYPE_ DEFAULT_NTG	The named tone generator configuration encapsulated in the **t_ve_ntg_algorithm_default_configuration_s** defined in Appendix B.14 on Page 275.

Attribute	Structure Association
VE_ATTRIBUTE_TYPE_ DEFAULT_NTD	The named tone detector configuration encapsulated in the **t_ve_ntd_algorithm_default_configuration_s** defined in Appendix B.13 on Page 274.
VE_ATTRIBUTE_TYPE_ DEFAULT_LINE_ECAN	The line echo canceller configuration encapsulated in the **t_ve_line_ecan_configuration_s** defined in Appendix B.10 on Page 264.
VE_ATTRIBUTE_TYPE_ CHANNEL_CONFIGURATION	The per-channel configuration for the VeriCall Edge system encapsulated in the **t_ve_channel_s** defined in Appendix B.21 on Page 289.
VE_ATTRIBUTE_TYPE_ USER_TRANSPORT_OPEN_FUNCTION	The function pointer of the function registered through the **VE_RegisterPacketTransportOpen** function.
VE_ATTRIBUTE_TYPE_ USER_TRANSPORT_SEND_FUNCTION	The function pointer of the function registered through the **VE_RegisterPacketTransportSend** function.
VE_ATTRIBUTE_TYPE_ USER_TRANSPORT_RECEIVE_FUNCTION	The function pointer of the function registered through the **VE_RegisterPacketTransportReceive** function.
VE_ATTRIBUTE_TYPE_ USER_TRANSPORT_CLOSE_FUNCTION	The function pointer of the function registered through the **VE_RegisterPacketTransportClose** function.
VE_ATTRIBUTE_TYPE_ USER_TRANSPORT_CONFIGURE_FUNCTION	The function pointer of the function registered through the **VE_RegisterPacketTransportConfigure** function.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_TRANSFER_OPEN_FUNCTION	The function pointers of the functions registered through the **VE_RegisterChannelDataSourceSinkOpen** function for both the receive and transmit.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_DATA_TRANSFER_FUNCTION	The function pointers of the functions registered through the **VE_RegisterChannelDataTransfer** function for both the receive and transmit.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_TRANSFER_CLOSE_FUNCTION	The function pointers of the functions registered through the **VE_RegisterChannelDataSourceSinkClose** function.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_TRANSFER_AVAIL_FUNCTION	The function pointers of the functions registered through the **VE_RegisterChannelDataTransferAvailability** function.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_TRANSFER_QUERY_FUNCTION	The function pointers of the functions registered through the **VE_RegisterChannelDataDeviceQuery** function.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_TRANSFER_CONFIG_FUNCTION	The function pointers of the functions registered through the **VE_RegisterChannelDataDeviceConfigure** function.
VE_ATTRIBUTE_TYPE_ READONLY_COMMIT_LOCK	This attribute will be set when a **VE_Commit** has been performed on the VeriCall Edge system.
VE_ATTRIBUTE_TYPE_ READONLY_INITIALIZE_LOCK	This attribute will be set when a **VE_Initialize** has been performed on the VeriCall Edge system.
VE_ATTRIBUTE_TYPE_ READONLY_USER	The users details of an associated channel encapsulated in the **t_ve_user_channel_details_s** defined in Appendix B.26 on Page 300.
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_SYSTEM_CALLS	The system calls statistics encapsulated in the **t_ve_stat_system_calls_s** defined in Appendix B.38.1 on Page 432.
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_VIDEO_ENCODER	The video encoder statistics encapsulated in the **t_ve_stat_channel_video_encoder_s** defined in Appendix B.38.3 on Page 433.
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_VIDEO_DECODER	The video decoder statistics encapsulated in the **t_ve_stat_channel_video_decoder_s** defined in Appendix B.38.4 on Page 434.

Attribute	Structure Association
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_RTCP	The RTCP statistics encapsulated in the **t_ve_stat_channel_rtcp_s** defined in Appendix B.38.5 on Page 435.
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_RTCP_PEER	The RTCP peer statistics encapsulated in the **t_ve_stat_channel_rtcp_peer_s** defined in Appendix B.38.6 on Page 438.
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_AV_SYNC	Audio visual synchronization statistics encapsulated in the **t_ve_stat_av_sync_s** defined in Appendix B.38.2.1 on Page 433.
VE_ATTRIBUTE_TYPE_ READONLY_USER_SIP_DETAILS	The users SIP details encapsulated in the **t_ve_sip_user_registration_s** defined in Appendix B.39 on Page 440.
VE_ATTRIBUTE_TYPE_READONLY_ DEFAULT_STREAM_STRUCTURE	The default stream parameters encapsulated in the **t_ve_stream_configuration_s** defined in Appendix B.24 on Page 298.
VE_ATTRIBUTE_TYPE_READONLY_ LAST_START_STREAM_STRUCTURE	Copy of the last stream parameter structure used. This is a read-only element and is encapsulated in the **t_ve_last_stream_configuration_s** defined in Appendix B.5 on Page 257.
VE_ATTRIBUTE_TYPE_ DEFAULT_DATA_END_POINT	The data-end point configuration parameters encapsulated in the **t_ve_data_end_point_default_configuration_s** defined in Appendix B.19 on Page 287.
VE_ATTRIBUTE_TYPE_ DEFAULT_TX_LIP_SYNC	The default transmit lip-synchronization settings encapsulated in the **t_ve_tx_lip_sync_default_configuration_s** defined in Appendix B.18 on Page 287.
VE_ATTRIBUTE_TYPE_ READONLY_USER_H323_DETAILS	The H.323 users details encapsulated in the **t_ve_h323_user_registration_s** defined in Appendix B.52 on Page 467.
VE_ATTRIBUTE_TYPE_ DEFAULT_H323_CONFIGURATION	The H.323 settings used by the VeriCall Edge system encapsulated in the **t_ve_h323_default_configuration_s** defined in Appendix B.58 on Page 475.
VE_ATTRIBUTE_TYPE_ SYSTEM_LOGGING	The level of system logging that should be performed. The level defaults to no logging and is encapsulated in the **t_ve_system_logging_s** defined in Appendix B.4 on Page 256.
VE_ATTRIBUTE_TYPE_ DEFAULT_ACOUSTIC_ECAN	The acoustic ECAN settings used by the VeriCall Edge channel encapsulated in the **t_ve_acoustic_ecan_configuration_s** defined in Appendix B.11 on Page 266.
VE_ATTRIBUTE_TYPE_ VIDEO_TEAR_PROTECT	The default enable/disable video tear protect settings encapsulated in the **t_ve_video_tear_protect_default_configuration_s** defined in Appendix B.20 on Page 288.
VE_ATTRIBUTE_TYPE_ DEFAULT_NAT_CONFIGURATION	The default network module configuration encapsulated in the **t_ve_nat_default_configuration_s** defined in Appendix B.60 on Page 491.
VE_ATTRIBUTE_TYPE_READONLY_ USER_SIP_PRESENCE_DETAILS	The users SIP message waiting details encapsulated in the **t_ve_sip_presence_data_s** defined in Appendix B.46 on Page 452.
VE_ATTRIBUTE_TYPE_READONLY_ USER_SIP_MW_DETAILS	The users SIP presence details encapsulated in the **t_ve_sip_mw_data_s** defined in Appendix B.47 on Page 457.
VE_ATTRIBUTE_TYPE_ DEFAULT_NET_CONFIGURATION	The default Network Services module configuration settings encapsulated in the **t_ve_net_default_configuration_s** defined in Appendix B.65 on Page 499.

Attribute	Structure Association
VE_ATTRIBUTE_TYPE_ DEFAULT_GAIN_CONTROL_TX	The gain control configuration for the channel transmit algorithm encapsulated in the t_ve_gain_control_configuration_s structure, defined in Appendix B.12 on Page 273
VE_ATTRIBUTE_TYPE_ DEFAULT_GAIN_CONTROL_RX	The gain control configuration for the channel receive algorithm encapsulated in the t_ve_gain_control_configuration_s structure, defined in Appendix B.12 on Page 273
VE_ATTRIBUTE_TYPE_ NUMBER_OF_ATTRIBUTES	Maximum number of attribute types – this is used internally by VeriCall Edge and should not be accessed by the user application.

Table C.1 and Table C.2 in Appendix C on Page 659 give details of the structures and index values used for the VeriCall Edge system attributes.

B.72 t_ve_return_code_e

The **t_ve_return_code_e** enumeration is used by all VeriCall Edge API functions to indicate the success or failure of a particular function call. The enumeration is defined in **ve_error.h** and is described below:

```
typedef enum{
VE_RETURN_CODE_SUCCESS                              = 0x0000,

/* Initialization and commit */
VE_RETURN_CODE_ALREADY_INITIALIZED                  = 0x0001,
VE_RETURN_CODE_UNINITIALIZED                        = 0x0002,
VE_RETURN_CODE_SIP_INITIALIZE_ERROR                 = 0x0003,
VE_RETURN_CODE_INVALID_ALG_CONFIG                   = 0x0004,
VE_RETURN_CODE_SCHEDULER_CONFIG_FAIL                = 0x0005,
VE_RETURN_CODE_QUEUE_ALLOCATION_ERROR               = 0x0006,
VE_RETURN_CODE_CALLBACKS_NOT_REGISTERED             = 0x0007,
VE_RETURN_CODE_INSUFFICIENT_CONTEXT_SPACE           = 0x0008,
VE_RETURN_CODE_ALREADY_COMMITTED                    = 0x0009,
VE_RETURN_CODE_NOT_COMMITED                         = 0x000A,
VE_RETURN_CODE_COMMIT_FAILED                        = 0x000B,
VE_RETURN_CODE_H323_INITIALIZE_ERROR                = 0x000C,
VE_RETURN_CODE_NAT_INITIALIZE_ERROR                 = 0x000D,
/* Default configuration errors */
VE_RETURN_CODE_ECAN_CONFIG_FAIL                     = 0x0100,
VE_RETURN_CODE_AJBM_CONFIG_FAIL                     = 0x0101,
VE_RETURN_CODE_PRE_POST_PROC_CONFIG_FAIL            = 0x0102,
VE_RETURN_CODE_SIP_CONFIG_FAIL                      = 0x0105,
VE_RETURN_CODE_NTG_CONFIG_FAIL                      = 0x0106,
VE_RETURN_CODE_NTD_CONFIG_FAIL                      = 0x0107,
VE_RETURN_CODE_CN_CONFIG_FAIL                       = 0x0108,
VE_RETURN_CODE_G711U_CONFIG_FAIL                    = 0x0109,
VE_RETURN_CODE_G711A_CONFIG_FAIL                    = 0x010A,
VE_RETURN_CODE_G729_CONFIG_FAIL                     = 0x010B,
VE_RETURN_CODE_TE_CONFIG_FAIL                       = 0x010C,
VE_RETURN_CODE_DEFAULT_CHANNEL_CONFIG_FAIL          = 0x010D,
VE_RETURN_CODE_IP_CONFIG_FAIL                       = 0x010E,
VE_RETURN_CODE_H263_CONFIG_FAIL                     = 0x010F,
VE_RETURN_CODE_GSM_AMR_CONFIG_FAIL                  = 0x0110,
VE_RETURN_CODE_DATA_END_POINT_CONFIG                = 0x0111,
VE_RETURN_CODE_H323_CONFIG_FAIL                     = 0x0112,
VE_RETURN_CODE_TX_LIP_SYNC_CONFIG_FAIL              = 0x0113,
VE_RETURN_CODE_G723_CONFIG_FAIL                     = 0x0114,
VE_RETURN_CODE_LOGGING_CONFIG_FAIL                  = 0x0115,
VE_RETURN_CODE_STATISTICS_CONFIG_FAIL               = 0x0116,
VE_RETURN_CODE_TEAR_PROTECT_CONFIG_FAIL             = 0x0117,
VE_RETURN_CODE_NAT_CONFIG_FAIL                      = 0x0118,
VE_RETURN_CODE_NAT_NETWORK_ERROR                    = 0x0119,
```

Continued >>

```
VE_RETURN_CODE_NAT_NETWORK_TIMEOUT              = 0x0120,
VE_RETURN_CODE_NAT_CANCELLED                    = 0x0121,
VE_RETURN_CODE_MPEG4_CONFIG_FAIL                = 0x0122,
VE_RETURN_CODE_G726_CONFIG_FAIL                 = 0x0123,
VE_RETURN_CODE_LAST_STREAM_INITIALIZATION_FAILED= 0x0124,
VE_RETURN_CODE_DEFAULT_STREAM_STRUCTURE_FAILED  = 0x0125,
VE_RETURN_CODE_ILBC_CONFIG_FAIL                 = 0x0126,
VE_RETURN_CODE_GAIN_CONTROL_CONFIG_FAIL         = 0x0127,

/* Function parameter errors */
VE_RETURN_CODE_NULL_POINTER                   = 0x0200,
VE_RETURN_CODE_INVALID_HANDLE                 = 0x0201,
VE_RETURN_CODE_INVALID_APPLICATION            = 0x0202,
VE_RETURN_CODE_INVALID_USER                   = 0x0203,
VE_RETURN_CODE_INVALID_ATTRIBUTE_INDEX        = 0x0204,
VE_RETURN_CODE_TOO_MANY_DIGITS_IN_STRING      = 0x0205,
VE_RETURN_CODE_TONE_NOT_SUPPORTED             = 0x0206,

VE_RETURN_CODE_STATS_TYPE_UNKNOWN                     = 0x0207,
VE_RETURN_CODE_STATS_TYPE_CANNOT_BE_RESET             = 0x0208,
VE_RETURN_CODE_INVALID_AUDIO_CONTROL_TYPE             = 0x0209,
VE_RETURN_CODE_INVALID_AUDIO_CONTROL_DIRECTION        = 0x020A,
VE_RETURN_CODE_INVALID_CHANNEL_CONFIGURATION          = 0x020B,
VE_RETURN_CODE_INVALID_AUDIO_FORMAT                   = 0x020C,
VE_RETURN_CODE_STORE_ENTRY_READ_ONLY                  = 0x020D,
VE_RETURN_CODE_USER_ALREADY_KNOWN                     = 0x020E,
VE_RETURN_CODE_INVALID_VIDEO_CONTROL_TYPE             = 0x020F,
VE_RETURN_CODE_INVALID_VIDEO_CONTROL_FILTER_SELECTION = 0x0210,
VE_RETURN_CODE_INVALID_VIDEO_CONTROL_DISPLAY_STATE     = 0x0211,
VE_RETURN_CODE_INVALID_VIDEO_CONTROL_SCALE_FACTOR     = 0x0212,
VE_RETURN_CODE_INVALID_VIDEO_CONTROL_ROTATION_FACTOR  = 0x0213,
VE_RETURN_CODE_INVALID_VIDEO_CONTROL_MIRROR_STATE     = 0x0214,
VE_RETURN_CODE_INVALID_VIDEO_CONTROL_MUTE_MODE        = 0x0215,
VE_RETURN_CODE_INVALID_VIDEO_CONTROL_ENCODE_FORMAT     = 0x0216,
VE_RETURN_CODE_INVALID_VIDEO_CONTROL_DATA_POINT_CONFIG = 0x0217,
VE_RETURN_CODE_INVALID_VIDEO_CONTROL_IMAGE_FORMAT     = 0x0218,
VE_RETURN_CODE_INVALID_VIDEO_CONTROL_LIP_SYNC_ADJUSTMENT = 0x0219,
VE_RETURN_CODE_INVALID_VIDEO_CONTROL_RESET_CONFIG_TYPE
                                                      = 0x021A,
VE_RETURN_CODE_CAPABILITY_MISMATCH                    = 0x021B,
VE_RETURN_CODE_ATTRIBUTE_OUT_OF_RANGE_ERROR           = 0x021C,
VE_RETURN_CODE_MEDIA_EXCEEDS_PACKET_BUDGET            = 0x021D,
VE_RETURN_CODE_CODEC_CONFIG_INCOMPATIBLE_PRE_PROCESSOR = 0x021E,
VE_RETURN_CODE_ILLEGAL_SCALING_FACTOR                 = 0x021F,
VE_RETURN_CODE_INVALID_ATTRIBUTE_SIZE                 = 0x0220,
VE_RETURN_CODE_STORE_ENTRY_WRITE_ONLY                 = 0x0221,
VE_RETURN_CODE_CHANNEL_OUT_OF_RANGE_ERROR             = 0x0222,
VE_RETURN_CODE_INVALID_VIDEO_CONTROL_INVALID_DATA_END_POINT
                                                      = 0x0230,
VE_RETURN_CODE_INVALID_NETWORK_TOS_CONFIGURATION      = 0x0240,
VE_RETURN_CODE_INVALID_TRANSMISSION_MEDIA_TYPE        = 0x0251,
VE_RETURN_CODE_BEEP_SELECTION_NOT_SUPPORTED           = 0x0252,
VE_RETURN_CODE_SIP_ADD_USER_REGISTRATION_DATA_FAILED  = 0x0260,
```

Continued >>

```
VE_RETURN_CODE_SIP_GET_USER_REGISTRATION_DATA_FAILED  = 0x0261,
VE_RETURN_CODE_SIP_UPDATE_USER_REGISTRATION_DATA_FAILED = 0x0262,
VE_RETURN_CODE_SIP_REMOVE_USER_REGISTRATION_DATA_FAILED
                                                      = 0x0263,
VE_RETURN_CODE_SIP_ADD_USER_PRESENCE_DATA_FAILED      = 0x0264,
VE_RETURN_CODE_SIP_UPDATE_USER_PRESENCE_DATA_FAILED    = 0x0265,
VE_RETURN_CODE_SIP_REMOVE_USER_PRESENCE_DATA_FAILED    = 0x0266,
VE_RETURN_CODE_SIP_GET_USER_PRESENCE_DATA_FAILED      = 0x0267,
VE_RETURN_CODE_SIP_INVALID_CALL_HOLD_METHOD           = 0x0268,
VE_RETURN_CODE_SIP_INVALID_MW_START_FLAG              = 0x0269,
VE_RETURN_CODE_H323_ADD_USER_REGISTRATION_DATA_FAILED = 0x0270,
VE_RETURN_CODE_H323_UPDATE_USER_REGISTRATION_DATA_FAILED = 0x0271,
VE_RETURN_CODE_H323_REMOVE_USER_REGISTRATION_DATA_FAILED
                                                      = 0x0272,
/* Memory based errors */
VE_RETURN_CODE_MEMORY_ALLOC_FAIL                      = 0x0300,
VE_RETURN_CODE_MEMORY_DECALLOC_FAIL                   = 0x0301,
VE_RETURN_CODE_USER_ALLOC_TOO_SMALL                   = 0x0302,
VE_RETURN_CODE_CORRUPT_POINTER                        = 0x0303,

/* Message based errors */
VE_RETURN_CODE_NO_MESSAGES                            = 0x0400,
VE_RETURN_CODE_FAILED_TO_SEND_MESSAGE                 = 0x0401,
VE_RETURN_CODE_MESSAGE_TOO_BIG                        = 0x0402,

/* Generic API errors */
VE_RETURN_CODE_RESOURCE_QUERY_FAIL                    = 0x0500,
VE_RETURN_CODE_API_NOT_SUPPORTED_ON_PLATFORM          = 0x0501,
VE_RETURN_CODE_FUNCTIONALITY_ALREADY_ACTIVATED        = 0x0502,
VE_RETURN_CODE_FUNCTIONALITY_ALREADY_DEACTIVATED      = 0x0503,
VE_RETURN_CODE_ILLEGAL_PARAMETER                      = 0x0504,
VE_RETURN_CODE_INVALID_STATE                          = 0x0505,

/* Internal Configuration store errors */
VE_RETURN_CODE_NO_STORE_ENTRY                         = 0x0600,
VE_RETURN_CODE_CHANNEL_NOT_AVAILABLE                  = 0x0601,
VE_RETURN_CODE_MEDIA_PROFILE_NOT_FOUND                = 0x0602,
VE_RETURN_CODE_LINE_ECAN_ID_NOT_FOUND                 = 0x0603,
VE_RETURN_CODE_ACOUSTIC_ECAN_ID_NOT_FOUND             = 0x0604,

/* Internal errors */
VE_RETURN_CODE_DTMF_CONSTRUCTION_FAIL                 = 0x0700,
VE_RETURN_CODE_INTERNAL_ERROR                         = 0x0701,

/* Message execution return codes */
VE_RETURN_CODE_CHANNEL_VAD_AND_CNG_CONFIG_FAILED      = 0x0800,
VE_RETURN_CODE_START_STREAM_FAILED                    = 0x0801,
VE_RETURN_CODE_START_STREAM_INACTIVE                  = 0x0802,
VE_RETURN_CODE_START_STREAM_DUPLICATED                = 0x0803,
VE_RETURN_CODE_STOP_STREAM_FAILED                     = 0x0804,
VE_RETURN_CODE_TONE_GENERATE_FAILED                   = 0x0805,
VE_RETURN_CODE_PLAY_PROMPT_FAILED                     = 0x0806,
VE_RETURN_CODE_RECORD_PROMPT_FAILED                   = 0x0807,
```

Continued >>

```
/* Fatal errors */
VE_RETURN_CODE_FATAL_ERROR_CHANNEL_CREATE              = 0x0900,
VE_RETURN_CODE_FATAL_ERROR_ILLEGAL_AJBM_CONFIG         = 0x0901

}t_ve_return_code_e;
```

A description of each return code is provided in Table A.1 on Page 245.

B.73 t_ve_verbose_level_e

The **t_ve_verbose_level_e** enumeration is used by the **t_ve_system_logging_s** structure, as defined in Appendix B.4 on Page 256. The **t_ve_verbose_level_e** enumeration allows the verbose level of the VeriCall Edge system logging to be defined. This enumeration is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_VERBOSE_LEVEL_DISABLED=0,
  VE_VERBOSE_LEVEL_FATAL,
  VE_VERBOSE_LEVEL_ERROR,
  VE_VERBOSE_LEVEL_WARNING,
  VE_VERBOSE_LEVEL_DEBUG,
  VE_VERBOSE_LEVEL_TRACE,
  VE_NUM_VERBOSE_LEVELS
}t_ve_verbose_level_e;
```

The enumeration elements of **t_ve_verbose_level_e** are as follows:

Element	Definition
VE_VERBOSE_LEVEL_DISABLED	All logging is disabled.
VE_VERBOSE_LEVEL_FATAL	Log all fatal messages.
VE_VERBOSE_LEVEL_ERROR	Log all error messages and all those of a lower verbosity level (fatal).
VE_VERBOSE_LEVEL_WARNING	Log all warning messages and all those of a lower verbosity level (error and fatal).
VE_VERBOSE_LEVEL_DEBUG	Log all debug messages and all those of a lower verbosity level (warning, error and fatal).
VE_VERBOSE_LEVEL_TRACE	Log all trace messages and all those of a lower verbosity level (debug, warning, error and fatal).
VE_NUM_VERBOSE_LEVELS	The number of verbose level. Only used by the VeriCall Edge system to determine the number of verbose levels within this enumeration.

B.74 t_ve_system_logging_destination_e

The **t_ve_system_logging_destination_e** enumeration is used within the **t_ve_system_logging_s** structure, as described in Appendix B.4 on Page 256. This enumeration type is used to determine the destination of the VeriCall Edge logging. This enumeration is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_SYSTEM_LOGGING_DESTINATION_DISABLED=0,
  VE_SYSTEM_LOGGING_DESTINATION_SYSLOG,
  VE_SYSTEM_LOGGING_DESTINATION_FILEHANDLE,
  VE_NUM_SYSTEM_LOGGING_DESTINATIONS,
}t_ve_system_logging_destination_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SYSTEM_LOGGING_DESTINATION_DISABLED	Disable logging.
VE_SYSTEM_LOGGING_DESTINATION_SYSLOG	Direct VeriCall Edge logging to *syslog*. This mode of operation is the default mode.
VE_SYSTEM_LOGGING_DESTINATION_FILEHANDLE	Direct VeriCall Edge logging to the specified file handle.
VE_NUM_SYSTEM_LOGGING_DESTINATIONS	The number of logging destination elements. Only used by the VeriCall Edge system and should not be used by the user application.

B.75 t_ve_system_logging_method_e

The **t_ve_system_logging_method_e** enumeration is used within the **t_ve_system_logging_s** structure, as described in Appendix B.4 on Page 256. This enumeration type is used to determine the method of VeriCall Edge logging to be used. This enumeration is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_SYSTEM_LOGGING_METHOD_DISABLED=0,
  VE_SYSTEM_LOGGING_METHOD_QUEUE,
  VE_SYSTEM_LOGGING_METHOD_DIRECT,
  VE_NUM_SYSTEM_LOGGING_METHODS,
}t_ve_system_logging_method_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SYSTEM_LOGGING_METHOD_DISABLED	Disable logging.
VE_SYSTEM_LOGGING_METHOD_QUEUE	VeriCall Edge logging should be queued before being sent to the specified logging destination. This will prevent any degradation to the performance of VeriCall Edge but may result in some messages being lost.
VE_SYSTEM_LOGGING_METHOD_DIRECT	VeriCall Edge logging should be sent directly to the specified logging destination.
VE_NUM_SYSTEM_LOGGING_METHODS	The number of logging method elements. Only used by the VeriCall Edge system and should not be used by the user application.

B.76 t_ve_signaling_type_e

The **t_ve_signaling_type_e** enumeration is used by **VE_GenerateDigitString** to determine the signaling method of the generated digit string. This enumeration is defined in **ve_tones.h** and is described below:

```
typedef enum{
  VE_DTMF_SIGNALING = 0x01
}t_ve_signaling_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_DTMF_SIGNALING	The digit string signaling type is DTMF.

B.77 t_ve_tone_transmission_media_type_e

The **t_ve_tone_transmission_media_type_e** enumeration is used by **VE_GenerateDigitString** to determine the transmission method of the generated digit strings. This enumeration is defined in **ve_tones.h** and is described below:

```
typedef enum{
  VE_TONE_TRANSMISSION_MEDIA_TYPE_PACKET_RTP          = 0x0001,
  VE_TONE_TRANSMISSION_MEDIA_TYPE_PACKET_RFC2833      = 0x0002,
  VE_TONE_TRANSMISSION_MEDIA_TYPE_AUDIO_LINEAR_PCM     = 0x0004,
  VE_TONE_TRANSMISSION_MEDIA_TYPE_PACKET_AUTO_SELECT = 0x0008
}t_ve_tone_transmission_media_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_TONE_TRANSMISSION_MEDIA_TYPE_PACKET_RTP	Transmit all tones in-band.
VE_TONE_TRANSMISSION_MEDIA_TYPE_PACKET_RFC2833	Transmit all tones out-of-band (DTMF Relay).
VE_TONE_TRANSMISSION_MEDIA_TYPE_AUDIO_LINEAR_PCM	Relay all tones as linear PCM.
VE_TONE_TRANSMISSION_MEDIA_TYPE_PACKET_AUTO_SELECT	Automatically selects media type to be used. If Telephony Events (TE) have been negotiated, RFC2833 would be chosen, otherwise in-band would be used.

If the user application does not track the SIP negotiation, then it is recommended to use the automatic media selection enumerated element (0x08). .

The tone transmission enumerated types can be OR'd together to allow bi-directional transmission of the digit string. Refer to the functions example code to see an instance of this.

B.78 t_ve_tone_type_e

The **t_ve_tone_type_e** enumeration is used by **VE_GenerateNamedTone** to determine the tone to generate. This enumeration is defined in **ve_tones.h** and is described below.

```
typedef enum{
  VE_TONE_NONE               = 0x0000,
  VE_TONE_RINGBACK           = 0x0101,
  VE_TONE_DIALTONE           = 0x0102,
  VE_TONE_SPECIALINFO        = 0x0103,
  VE_TONE_BUSYTONE           = 0x0104,
  VE_TONE_CONGESTION_TONE    = 0x0105,
  VE_TONE_2100_TONE          = 0x0106,
  VE_TONE_1100_TONE          = 0x0107
}t_ve_tone_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_TONE_NONE	Stop all tones.
VE_TONE_RINGBACK	Tone generation of Ring Back.
VE_TONE_DIALTONE	Tone generation of Dial Tone.
VE_TONE_SPECIALINFO	Tone generation of Special Information tone.
VE_TONE_BUSYTONE	Tone generation of Busy tone.
VE_TONE_CONGESTION_TONE	Tone generation of Congestion tone.
VE_TONE_2100_TONE	Tone generation of 2100Hz.
VE_TONE_1100_TONE	Tone generation of 1100Hz.

B.79 t_ve_tone_direction_e

The **t_ve_tone_direction_e** enumeration is used by **VE_GenerateNamedTone** and **VE_GenerateUserTone** to determine the transmission media stream direction that the tone should be played into. This enumeration is defined in **ve_tones.h** and is described below:

```
typedef enum{
  VE_TONE_DIRECTION_LOCAL  = 0x0001,
  VE_TONE_DIRECTION_REMOTE = 0x0002,
}t_ve_tone_direction_e;
```

The enumeration elements are as follows:

Element	Definition
VE_TONE_DIRECTION_LOCAL	Transmit all tones locally, meaning all tones will be heard by the local-end user.
VE_TONE_DIRECTION_REMOTE	Transmit all tones remotely, meaning all tones will be head by the far-end user.

The tone direction enumerated types can be OR'd together to allow bi-directional transmission of the digit string. Refer to the functions example code to see an instance of this.

B.80 t_ve_country_e

The **t_ve_country_e** enumeration is used by **VE_GenerateNamedTone** to determine the country type of tone to generate. This enumeration is defined in **ve_tones.h** and is described below:

```
typedef enum{
  VE_COUNTRY_TYPE_NOT_APPLICABLE,
  VE_COUNTRY_TYPE_UK,
  VE_COUNTRY_TYPE_USA,
  VE_COUNTRY_TYPE_JAPAN,
  VE_COUNTRY_TYPE_FRANCE,
  VE_COUNTRY_TYPE_CHINA,
  VE_COUNTRY_TYPE_RUSSIA,
  VE_COUNTRY_TYPE_KOREA
}t_ve_country_e;
```

The enumeration elements are as follows:

Element	Definition
VE_COUNTRY_TYPE_NOT_APPLICABLE	There is no country specific tone to be generated.
VE_COUNTRY_TYPE_UK	Tones generated will be UK tones.
VE_COUNTRY_TYPE_USA	Tones generated will be USA tones.
VE_COUNTRY_TYPE_JAPAN	Tones generated will be Japanese tones.
VE_COUNTRY_TYPE_FRANCE	Tones generated will be French tones.
VE_COUNTRY_TYPE_CHINA	Tones generated will be Chinese tones.
VE_COUNTRY_TYPE_RUSSIA	Tones generated will be Russian tones.
VE_COUNTRY_TYPE_KOREA	Tones generated will be Korean tones.

B.81 t_ve_ip_default_configuration_e

The **t_ve_ip_default_configuration_e** enumeration is part of the **t_ve_ip_default_configuration_s** structure as described in Appendix B.7 on Page 260. The **t_ve_ip_default_configuration_e** enumeration type contains the type of IP configuration to be used by the VeriCall Edge system. The **t_ve_ip_default_configuration_e** enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_IP_DEFAULT_CONFIGURATION_VERSION_V4,
  VE_IP_DEFAULT_CONFIGURATION_FQDN,
}t_ve_ip_default_configuration_e;
```

The enumeration elements are as follows:

Element	Definition
VE_IP_DEFAULT_CONFIGURATION_VERSION_V4	The IP address is an IPv4 address type.
VE_IP_DEFAULT_CONFIGURATION_FQDN	The IP address is a fully qualified domain name.

B.82 t_ve_ip_port_reopen_e

The **t_ve_ip_port_reopen_e** enumeration is part of the **t_ve_ip_default_configuration_s** structure as described in Appendix B.7 on Page 260. The **t_ve_ip_default_configuration_e** enumeration type determines whether the IP port configuration is capable of being re-opened or not. The **t_ve_ip_default_configuration_e** enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_IP_PORT_REOPEN_FALSE,
  VE_IP_PORT_REOPEN_TRUE,
}t_ve_ip_port_reopen_e;
```

The enumeration elements are as follows:

Element	Definition
VE_IP_PORT_REOPEN_FALSE	The ability to re-open the IP port is false.
VE_IP_PORT_REOPEN_TRUE	The ability to re-open the IP port is true.

B.83 t_ve_process_selector_default_configuration_e

The **t_ve_process_selector_default_configuration_e** enumeration is part of the **t_ve_pre_proc_default_configuration_s** structure, as described in Appendix B.15 on Page 275, and the **t_ve_post_proc_default_configuration_s** structure, as described in Appendix B.16 on Page 280. The **t_ve_process_selector_default_configuration_e** enumeration type contains the type of pre and post processing should be applied to the specified channel. The **t_ve_process_selector_default_configuration_e** enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_PROCESS_SELECTOR_DEFAULT_NULL = 0,
  VE_PROCESS_SELECTOR_DEFAULT_G711,
  VE_PROCESS_SELECTOR_DEFAULT_VIDEO
}t_ve_process_selector_default_configuration_e;
```

The enumeration elements are as follows:

Element	Definition
VE_PROCESS_SELECTOR_DEFAULT_NULL	There is no requirement for a pre or post-processor.
VE_PROCESS_SELECTOR_DEFAULT_G711	The pre or post-processor is for audio.
VE_PROCESS_SELECTOR_DEFAULT_VIDEO	The pre or post-processor is for video.

B.84 t_ve_default_configuration_boolean_e

The **t_ve_default_configuration_boolean_e** enumeration is part of various default configuration structures to enable or disable configuration components. The **t_ve_default_configuration_boolean_e** enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_DEFAULT_CONFIGURATION_FALSE  = 0,
  VE_DEFAULT_CONFIGURATION_TRUE   = 1
}t_ve_default_configuration_boolean_e;
```

The enumeration elements are as follows:

Element	Definition
VE_DEFAULT_CONFIGURATION_FALSE	Used to de-activate (disable) the configuration component.
VE_DEFAULT_CONFIGURATION_TRUE	Used to activate (enable) the configuration component.

B.85 t_ve_g711_default_mode_e

The **t_ve_g711_default_mode_e** enumeration is part of the G.711 default configuration structures, **t_ve_g711_pre_proc_default_configuration_s** defined in Appendix B.15.2 on Page 277 and **t_ve_g711_post_proc_default_configuration_s** defined in Appendix B.16.2 on Page 281. The **t_ve_g711_default_mode_e** enumeration determines the G.711 data format to be used. The **t_ve_g711_default_mode_e** enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_G711_DEFAULT_MODE_ULAW = 0,
  VE_G711_DEFAULT_MODE_ALAW,
}t_ve_g711_default_mode_e;
```

The enumeration elements are as follows:

Element	Definition
VE_G711_DEFAULT_MODE_ULAW	Define the G.711 default mode of operation as μ-Law.
VE_G711_DEFAULT_MODE_ALAW	Define the G.711 default mode of operation as a-Law.

B.86 t_ve_line_16ms_ecan_nlp_default_settings_e

The **t_ve_line_16ms_ecan_nlp_default_settings_e** enumeration is part of the **t_ve_line_ecan_16ms_ecan_configuration_s** structure defined in Appendix B.10.1.1 on Page 265. The **t_ve_line_16ms_ecan_nlp_default_settings_e** enumeration determines the NLP settings to be utilized by the VeriCall Edge ECAN. The enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_LINE_16MS_ECAN_NLP_DEFAULT_CONFIGURATION_SETTINGS_SOFT,
  VE_LINE_16MS_ECAN_NLP_DEFAULT_CONFIGURATION_SETTINGS_NORMAL,
  VE_LINE_16MS_ECAN_NLP_DEFAULT_CONFIGURATION_SETTINGS_HARSH,
}t_ve_line_16ms_ecan_nlp_default_settings_e;
```

The enumeration elements of **t_ve_line_16ms_ecan_nlp_default_settings_e** are as follows:

Element	Definition
VE_LINE_16MS_ECAN_NLP_DEFAULT_ CONFIGURATION_SETTINGS_SOFT	Define the ECAN NLP settings as being “soft”, allowing a decrease in the amount of cancelled echo.
VE_LINE_16MS_ECAN_NLP_DEFAULT_ CONFIGURATION_SETTINGS_NORMAL	Define the ECAN NLP settings as being “normal”, allowing an acceptable amount of echo cancellation.
VE_LINE_16MS_ECAN_NLP_DEFAULT_ CONFIGURATION_SETTINGS_HARSH	Define the ECAN NLP settings as being “harsh”, allowing an increase in the amount of cancelled echo.

For optimal use of this enumerated type with the VeriCall Edge system, Trinity Convergence recommends that the “normal” NLP setting be always used.

B.87 t_ve_acoustic_ecan_id_e

The **t_ve_acoustic_ecan_id_e** enumeration is part of the **t_ve_acoustic_ecan_configuration_s** structure defined in Appendix B.11 on Page 266. The **t_ve_acoustic_ecan_id_e** enumeration determines the VeriCall Edge acoustic ECAN type to be used. The enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  AEC_ACOUSTIC_ECAN_ID,
  AEC_COMPLEX_ACOUSTIC_ECAN_ID,
  AES_ACOUSTIC_ECAN_ID,
  MAX_ACOUSTIC_ECAN_ID
}t_ve_acoustic_ecan_id_e;
```

The enumeration elements are as follows:

Element	Definition
AEC_ACOUSTIC_ECAN_ID	Indicates that the Acoustic ECAN configuration will be used. This configuration is the **t_ve_acoustic_ecan_aec_configuration_s** structure as described in Appendix B.11.1.1 on Page 267.
AEC_COMPLEX_ACOUSTIC_ECAN_ID	Indicates that the complex Acoustic ECAN configuration will be used. This configuration is the **t_ve_acoustic_ecan_aec_complex_configuration_s** structure as described in Appendix B.11.1.2 on Page 268.
AES_ACOUSTIC_ECAN_ID	Indicates that the Acoustic Suppressor ECAN configuration will be used. This configuration is the **t_ve_acoustic_ecan_aes_configuration_s** structure as described in Appendix B.11.1.3 on Page 271.
MAX_LINE_ECAN_ID	The maximum number of line ECAN type ID's available and should be the last entry of the enumerated list.

B.88 t_ve_acoustic_ecan_aes_mode_e

The **t_ve_acoustic_ecan_aes_mode_e** enumeration is part of the **t_ve_acoustic_ecan_aes_configuration_s** structure as described in Appendix B.11.1.3 on Page 271. The **t_ve_acoustic_ecan_aes_mode_e** enumeration determines the VeriCall Edge acoustic echo suppressor mode of operation. The enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_ACOUSTIC_ECAN_AES_MODE_NORMAL,
  VE_ACOUSTIC_ECAN_AES_MODE_HIGH,
}t_ve_acoustic_ecan_aes_mode_e;
```

The enumeration elements are as follows:

Element	Definition
VE_ACOUSTIC_ECAN_AES_MODE_NORMAL	Mode of echo suppression operation for "normal" echo levels.
VE_ACOUSTIC_ECAN_AES_MODE_HIGH	Mode of echo suppression operation for "severe" echo levels.

In most subjectively normal situations, the normal echo suppression mode of operation is sufficient. In some extreme cases it may be necessary to switch the mode of operation to high.

B.89 t_ve_line_ecan_id_e

The **t_ve_line_ecan_id_e** enumeration is part of the **t_ve_line_ecan_configuration_s** structure defined in Appendix B.10 on Page 264. The **t_ve_line_ecan_id_e** enumeration determines the VeriCall Edge line ECAN to be used. The enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VC_16MS_LINE_ECAN_ID,
  MAX_LINE_ECAN_ID
}t_ve_line_ecan_id_e;
```

The enumeration elements are as follows:

Element	Definition
VE_16MS_LINE_ECAN_ID	Indicates that the 16ms Line ECAN configuration will be used. This configuration is the t_ve_line_ecan_16ms_ecan_configuration_s structure as described in Appendix B.10.1.1 on Page 265.
MAX_LINE_ECAN_ID	The maximum number of line ECAN type ID's available and should be the last entry of the enumerated list.

B.90 t_ve_ajbm_algorithm_default_lip_sync_state_e

The **t_ve_ajbm_algorithm_default_lip_sync_state_e** enumeration is part of the **t_ve_ajbm_algorithm_default_configuration_s** structure defined in Appendix B.9 on Page 262. The **t_ve_ajbm_algorithm_default_lip_sync_state_e** enumeration determines whether this jitter buffer has the ability to perform lip-synchronization between associated audio and video channels within the VeriCall Edge system. The enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_AJBM_ALGORITHM_LIP_SYNC_STATE_DISABLE = 0,
  VE_AJBM_ALGORITHM_LIP_SYNC_STATE_ENABLE
}t_ve_ajbm_algorithm_default_lip_sync_state_e;
```

The enumeration elements are as follows:

Element	Definition
VE_AJBM_ALGORITHM_LIP_SYNC_STATE_DISABLE	Disable the ability of this AJBM to perform lip-synchronization between associated audio and video channels.
VE_AJBM_ALGORITHM_LIP_SYNC_STATE_ENABLE	Enable the ability of this AJBM to perform lip-synchronization between associated audio and video channels.

B.91 t_ve_gain_control_mode_e

The **t_ve_gain_control_mode_e** enumeration is part of the **t_ve_gain_control_configuration_s** structure as defined in Appendix B.12 on Page 273. The enumeration determines the mode of gain control to be applied. The enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_GAIN_CONTROL_MODE_MANUAL       = 0,
  VE_GAIN_CONTROL_MODE_AUTOMATIC    = 1,
}t_ve_gain_control_mode_e;
```

The enumeration elements are as follows:

Element	Definition
VE_GAIN_CONTROL_MODE_MANUAL	Manual Gain Control will be applied.
VE_GAIN_CONTROL_MODE_AUTOMATIC	Automatic Gain Control will be applied.

B.92 t_ve_gain_control_agc_mode_e

The **t_ve_gain_control_agc_mode_e** enumeration is part of the **t_ve_gain_control_configuration_s** structure as defined in Appendix B.12 on Page 273. The enumeration determines the mode of Automatic Gain Control (AGC). The enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_GAIN_CONTROL_AGC_MODE_LOW      = 0,
  VE_GAIN_CONTROL_AGC_MODE_MEDIUM   = 1,
  VE_GAIN_CONTROL_AGC_MODE_HIGH     = 2,
}t_ve_gain_control_agc_mode_e;
```

The enumeration elements are as follows:

Element	Definition
VE_GAIN_CONTROL_AGC_MODE_LOW	AGC will maintain a low output level.
VE_GAIN_CONTROL_AGC_MODE_MEDIUM	AGC will maintain a medium output level.
VE_GAIN_CONTROL_AGC_MODE_HIGH	AGC will maintain a high output level.

B.93 t_ve_ntd_country_type_default_configuration_e

The **t_ve_ntd_country_type_default_configuration_e** enumeration is part of the **t_ve_ntd_algorithm_default_configuration_s** structure defined in Appendix B.11.1.3 on Page 271. The **t_ve_ntd_country_type_default_configuration_e** enumeration determines the country specific tones to be detected by the VeriCall Edge Named Tone Detector (NTD). The **t_ve_ntd_country_type_default_configuration_e** enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_NTD_DEFAULT_CONFIGURATION_UK = 1,
  VE_NTD_DEFAULT_CONFIGURATION_USA,
  VE_NTD_DEFAULT_CONFIGURATION_JAPAN,
  VE_NTD_DEFAULT_CONFIGURATION_FRANCE,
  VE_NTD_DEFAULT_CONFIGURATION_CHINA,
  VE_NTD_DEFAULT_CONFIGURATION_RUSSIA,
  VE_NTD_DEFAULT_CONFIGURATION_KOREA
}t_ve_ntd_country_type_default_configuration_e;
```

The enumeration elements are as follows:

Element	Definition
VE_NTD_DEFAULT_CONFIGURATION_UK	Define the NTD country configuration as being UK tones.
VE_NTD_DEFAULT_CONFIGURATION_USA	Define the NTD country configuration as being USA tones.
VE_NTD_DEFAULT_CONFIGURATION_JAPAN	Define the NTD country configuration as being Japanese tones.
VE_NTD_DEFAULT_CONFIGURATION_FRANCE	Define the NTD country configuration as being French tones.
VE_NTD_DEFAULT_CONFIGURATION_CHINA	Define the NTD country configuration as being Chinese tones.
VE_NTD_DEFAULT_CONFIGURATION_RUSSIA	Define the NTD country configuration as being Russian Tones.
VE_NTD_DEFAULT_CONFIGURATION_KOREA	Define the NTD country configuration as being Korean tones.

B.94 t_ve_video_default_local_image_enable_e

The **t_ve_video_default_local_image_enable_e** enumeration is part of the **t_ve_video_pre_proc_default_configuration_s** structure defined in Appendix B.15.3 on Page 277. The **t_ve_video_default_local_image_enable_e** enumeration determines whether the video image from the local camera is to be displayed to the defined local display. The enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_VIDEO_DEFAULT_LOCAL_IMAGE_ENABLE_OFF = 0,
  VE_VIDEO_DEFAULT_LOCAL_IMAGE_ENABLE_ON
}t_ve_video_default_local_image_enable_e;
```

The enumeration elements are as follows:

Element	Definition
VE_VIDEO_DEFAULT_LOCAL_IMAGE_ENABLE_OFF	Deactivate the displaying of the local camera image to the defined local display.
VE_VIDEO_DEFAULT_LOCAL_IMAGE_ENABLE_ON	Activate the displaying of the local camera image to the defined local display.

B.95 t_ve_video_default_image_rotate_e

The **t_ve_video_default_image_rotate_e** enumeration is part of the **t_ve_video_pre_proc_default_configuration_s** structure defined in Appendix B.15.3 on Page 277. The **t_ve_video_default_image_rotate_e** enumeration determines the clockwise rotation to be performed on the image in degrees. The enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_VIDEO_DEFAULT_IMAGE_ROTATE_0_DEGREES = 0,
  VE_VIDEO_DEFAULT_IMAGE_ROTATE_90_DEGREES,
  VE_VIDEO_DEFAULT_IMAGE_ROTATE_180_DEGREES,
  VE_VIDEO_DEFAULT_IMAGE_ROTATE_270_DEGREES,
}t_ve_video_default_image_rotate_e;
```

The enumeration elements are as follows:

Element	Definition
VE_VIDEO_DEFAULT_IMAGE_ROTATE_0_DEGREES	No rotation is required of the current image.
VE_VIDEO_DEFAULT_IMAGE_ROTATE_90_DEGREES	Rotate the image clockwise by 90°.
VE_VIDEO_DEFAULT_IMAGE_ROTATE_180_DEGREES	Rotate the image clockwise by 180°.
VE_VIDEO_DEFAULT_IMAGE_ROTATE_270_DEGREES	Rotate the image clockwise by 270°.

B.96 t_ve_video_default_image_mirror_e

The **t_ve_video_default_image_mirror_e** enumeration is part of the **t_ve_video_pre_proc_default_configuration_s** structure defined in Appendix B.15.3 on Page 277. The **t_ve_video_default_image_mirror_e** enumeration determines the plane the image to be mirrored on. The enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_VIDEO_DEFAULT_IMAGE_MIRROR_DISABLE = 0,
  VE_VIDEO_DEFAULT_IMAGE_MIRROR_ENABLE_H_PLANE,
  VE_VIDEO_DEFAULT_IMAGE_MIRROR_ENABLE_V_PLANE,
  VE_VIDEO_DEFAULT_IMAGE_MIRROR_ENABLE_H_AND_V_PLANES;
}t_ve_video_default_image_mirror_e;
```

The enumeration elements are as follows:

Element	Definition
VE_VIDEO_DEFAULT_IMAGE_MIRROR_DISABLE	The image is not mirrored.
VE_VIDEO_DEFAULT_IMAGE_MIRROR_ENABLE_H_PLANE	The image is mirrored on the horizontal plane only.
VE_VIDEO_DEFAULT_IMAGE_MIRROR_ENABLE_V_PLANE	The image is mirrored on the vertical plane only.
VE_VIDEO_DEFAULT_IMAGE_MIRROR_ENABLE_H_AND_V_PLANES	The image is mirrored on the horizontal and vertical plane.

B.97 t_ve_video_color_format_e

The **t_ve_video_color_format_e** enumeration is part of the **t_ve_video_pre_proc_default_configuration_s** structure defined in Appendix B.15.3 on Page 277. The **t_ve_video_color_format_e** enumeration determines the supported image format. The enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_YUV420_COLOR_FORMAT = 0,
  VE_YUV422_COLOR_FORMAT,
  VE_RGB_16_BIT_COLOR_FORMAT,
  VE_RGB_32_BIT_COLOR_FORMAT,
  VE_YUV422_YUYV_COLOR_FORMAT,
  VE_YUV422_YVYU_COLOR_FORMAT,
  VE_YUV422_UYVY_COLOR_FORMAT,
  VE_YUV422_VYUY_COLOR_FORMAT
}t_ve_video_color_format_e;
```

The enumeration elements are as follows:

Element	Definition
VE_YUV420_COLOR_FORMAT	The supported image format is YU420.
VE_YUV422_COLOR_FORMAT	The supported image format is YU422.
VE_RGB_16_BIT_COLOR_FORMAT	The supported image format is 16-bit RGB.
VE_RGB_32_BIT_COLOR_FORMAT	The supported image format is 32-bit RGB.
VE_YUV422_YUYV_COLOR_FORMAT	The supported image format is YUV422 YUYV.
VE_YUV422_YVYU_COLOR_FORMAT	The supported image format is YUV422 YVYU.
VE_YUV422_UYVY_COLOR_FORMAT	The supported image format is YUV422 UYVY.
VE_YUV422_VYUY_COLOR_FORMAT	The supported image format is YUV422 VYUY.

B.98 t_ve_video_filter_selection_e

The **t_ve_video_filter_selection_e** enumeration is part of the **t_ve_video_pre_proc_default_configuration_s** structure defined in Appendix B.15.3 on Page 277. The **t_ve_video_filter_selection_e** enumeration determines the filter functionality to be applied to the current image. The enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_VIDEO_IMAGE_FILTER_SELECTION_ALL_DISABLE = 0,
  VE_VIDEO_IMAGE_FILTER_SELECTION_DERING_ENABLE,
  VE_VIDEO_IMAGE_FILTER_SELECTION_DEBLOCK_ENABLE,
  VE_VIDEO_IMAGE_FILTER_SELECTION_ALL_ENABLE,
}t_ve_video_filter_selection_e;
```

Continued >>

The enumeration elements of **t_ve_video_filter_selection_e** are as follows:

Element	Definition
VE_VIDEO_IMAGE_FILTER_SELECTION_ALL_DISABLE	The image is not filtered.
VE_VIDEO_IMAGE_FILTER_SELECTION_DERING_ENABLE	The de-ringing filter is enabled for the current image.
VE_VIDEO_IMAGE_FILTER_SELECTION_DEBLOCK_ENABLE	The de-blocking filter is enabled for the current image.
VE_VIDEO_IMAGE_FILTER_SELECTION_ALL_ENABLE	The de-ringing and de-blocking filter is enabled for the current image.

B.99 t_ve_data_end_point_default_type_e

The **t_ve_data_end_point_default_type_e** enumeration is used by the **t_ve_data_end_point_default_configuration_s** structure, as defined in Appendix B.19 on Page 287. The **t_ve_data_end_point_default_type_e** enumeration determines the type of data-end point to be used. The enumeration is defined in **ve_configuration.h** and is described below:

```
typedef enum{
  VE_DATA_END_POINT_DEFAULT_TYPE_AUDIO = 0,
  VE_DATA_END_POINT_DEFAULT_TYPE_VIDEO
}t_ve_data_end_point_default_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_DATA_END_POINT_DEFAULT_TYPE_AUDIO	The data-end point is an audio data-end point.
VE_DATA_END_POINT_DEFAULT_TYPE_VIDEO	The data-end point is a video data-end point.

B.100 t_ve_channel_data_format_e

The **t_ve_channel_data_format_e** enumeration is part of the **t_ve_channel_data_configuration_s** structure defined in Appendix B.21.1 on Page 290. The **t_ve_channel_data_format_e** enumeration determines the data format of the data-side channel configuration. The **t_ve_channel_data_format_e** enumeration is defined in **ve_channel.h** and is described below:

```
typedef enum{
  VE_CHANNEL_DATA_FORMAT_16BIT_PCM_LINEAR,
  VE_CHANNEL_DATA_FORMAT_G711,
  VE_CHANNEL_DATA_FORMAT_VIDEO
}t_ve_channel_data_format_e;
```

The enumeration elements are as follows:

Element	Definition
VE_CHANNEL_DATA_FORMAT_16BIT_PCM_LINEAR	The channel data format is 16-bit PCM linear.
VE_CHANNEL_DATA_FORMAT_G711	The channel data format is G.711 audio.
VE_CHANNEL_DATA_FORMAT_VIDEO	The channel data format is video.

B.101 t_ve_channel_config_boolean_e

The **t_ve_channel_config_boolean_e** enumeration is part of the **t_ve_channel_s** structure defined in Appendix B.21 on Page 289. The **t_ve_channel_config_boolean_e** enumeration is used to enable or disable channel configuration components. The **t_ve_channel_config_boolean_e** enumeration is defined in **ve_channel.h** and is described below:

```
typedef enum{
  VE_CHANNEL_CONFIGURATION_FALSE = 0,
  VE_CHANNEL_CONFIGURATION_TRUE = 1
}t_ve_channel_config_boolean_e;
```

The enumeration elements are as follows:

Element	Definition
VE_CHANNEL_CONFIGURATION_FALSE	Used to de-activate (disable) the specified channel.
VE_CHANNEL_CONFIGURATION_TRUE	Used to activate (enable) the specified channel.

B.102 t_ve_channel_type_config_e

The **t_ve_channel_type_config_e** enumeration is part of the **t_ve_channel_s** structure defined in Appendix B.21 on Page 289. The **t_ve_channel_type_config_e** enumeration determines the media type to be used within this channel. The enumeration is defined in **ve_channel.h** and is described below:

```
typedef enum{
  VE_CHANNEL_TYPE_CONFIG_AUDIO = 0,
  VE_CHANNEL_TYPE_CONFIG_VIDEO = 1
}t_ve_channel_type_config_e;
```

The enumeration elements are as follows:

Element	Definition
VE_CHANNEL_TYPE_CONFIG_AUDIO	The channel is configured for audio.
VE_CHANNEL_TYPE_CONFIG_VIDEO	The channel is configured for video.

B.103 t_ve_high_pass_filter_state_e

The **t_ve_high_pass_filter_state_e** enumeration is part of the **t_ve_channel_post_conditioning_s** structure defined in Appendix B.21.2.2 on Page 292. The **t_ve_high_pass_filter_state_e** enumeration determines whether the post-processor high-pass filter is to be enabled or not. The enumeration is defined in **ve_channel.h** and is described below:

```
typedef enum{
  VE_HIGH_PASS_FILTER_DISABLED = 0,
  VE_HIGH_PASS_FILTER_ENABLED
}t_ve_high_pass_filter_state_e;
```

The enumeration elements are as follows:

Element	Definition
VE_HIGH_PASS_FILTER_DISABLED	The post-processor high-pass filter is disabled.
VE_HIGH_PASS_FILTER_ENABLED	The post-processor high-pass filter is enabled.

This enumeration is reserved for future use and should not be used within this release of VeriCall Edge.

B.104 t_ve_media_data_type_e

The **t_ve_media_data_type_e** enumeration is part of the **t_ve_media_field_s** structure defined in Appendix B.28 on Page 317. The **t_ve_media_data_type_e** enumeration determines the media type used by a VeriCall Edge channel. The **t_ve_media_data_type_e** enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_DATA_TYPE_INVALID,
  VE_MEDIA_DATA_TYPE_AUDIO,
  VE_MEDIA_DATA_TYPE_VIDEO
}t_ve_media_data_type_e;
```

The enumeration element is as follows:

Element	Definition
VE_MEDIA_DATA_TYPE_INVALID	The media type used by the VeriCall Edge system is an invalid type.
VE_MEDIA_DATA_TYPE_AUDIO	The media type used by the VeriCall Edge system is of type AUDIO.
VE_MEDIA_DATA_TYPE_VIDEO	The media type used by the VeriCall Edge system is of type VIDEO.

B.105 t_ve_media_transport_type_e

The **t_ve_media_transport_type_e** enumeration is part of the **t_ve_media_field_s** structure defined in Appendix B.27.1 on Page 303. The **t_ve_media_transport_type_e** enumeration determines the packet-side transportation type used by a VeriCall Edge channel. The **t_ve_media_transport_type_e** enumeration is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_TRANSPORT_TYPE_RTP
}t_ve_media_transport_type_e;
```

The enumeration element is as follows:

Element	Definition
VE_TRANSPORT_TYPE_RTP	The transport type used by the VeriCall Edge system is of type RTP.

B.106 t_ve_media_type_e

The **t_ve_media_type_e** enumeration is part of the **t_ve_media_field_s** structure defined in Appendix B.27.1 on Page 303. The **t_ve_media_type_e** enumeration determines the media data format to be expected by the VeriCall Edge system from the registered data manager callback functions. The **t_ve_media_type_e** enumeration is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_GENERIC_MEDIA_TYPE_INVALID = 0,
  VE_GENERIC_MEDIA_TYPE_AUDIO,
  VE_GENERIC_MEDIA_TYPE_VIDEO
  VE_GENERIC_MEDIA_TYPE_NUM_SELECTIONS
}t_ve_media_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_GENERIC_MEDIA_TYPE_INVALID	The per-channel media type is of an invalid type.
VE_GENERIC_MEDIA_TYPE_AUDIO	The per-channel media type is an audio data format.
VE_GENERIC_MEDIA_TYPE_VIDEO	The per-channel media type is a video data format.
VE_GENERIC_MEDIA_TYPE_NUM_SELECTIONS	The number of entries in this enumerated type.

B.107 t_ve_media_ip_configuration_e

The **t_ve_media_ip_configuration_e** enumeration is part of the **t_ve_media_ip_configuration_s** structure defined in Appendix B.27.2 on Page 304. The **t_ve_media_ip_configuration_e** enumeration determines the IP configuration format and is defined in **ve_defs.h** as described below:

```
typedef enum{
  VE_MEDIA_IP_CONFIG_VERSION_V4,
  VE_MEDIA_IP_CONFIG_FQDN,
}t_ve_media_ip_configuration_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_IP_CONFIG_VERSION_V4	The IP configuration used by the VeriCall Edge system is of an IPv4 type.
VE_MEDIA_IP_CONFIG_FQDN	The IP configuration used by the VeriCall Edge system is a fully qualified domain name.

B.108 t_ve_media_g711_mode_e

The **t_ve_media_g711_mode_e** enumeration is part of the G.711 configuration structures, such as the **t_ve_media_g711_vad_encoder_algorithm_configuration_s** structure defined in Appendix B.29.1 on Page 317. The **t_ve_media_g711_mode_e** enumeration determines the G.711 data format to be used. The **t_ve_media_g711_mode_e** enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_G711_MODE_ULAW,
  VE_MEDIA_G711_MODE_ALAW,
  VE_MEDIA_G711_MODE_UNCHANGED
}t_ve_media_g711_mode_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_G711_MODE_ULAW	Define the G.711 mode of operation as μ-Law.
VE_MEDIA_G711_MODE_ALAW	Define the G.711 mode of operation as a-Law.
VE_MEDIA_G711_MODE_UNCHANGED	Define the G.711 mode of operation as being unchanged.

B.109 t_ve_media_vad_enable_boolean_e

The **t_ve_media_vad_enable_boolean_e** enumeration is part of vocoder configuration structures, such as the **t_ve_media_g711_vad_encoder_algorithm_configuration_s** or **t_ve_media_g723_1_encoder_algorithm_configuration_s** structures defined in Appendix B.29.1 on Page 317 and Appendix B.29.5 on Page 319. The **t_ve_media_vad_enable_boolean_e** enumeration determines whether the VAD capability of the vocoder configuration structure is enabled or disabled. The **t_ve_media_vad_enable_boolean_e** enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_VAD_ENABLE_BOOLEAN_FALSE,
  VE_MEDIA_VAD_ENABLE_BOOLEAN_TRUE
}t_ve_media_vad_enable_boolean_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_VAD_ENABLE_BOOLEAN_FALSE	Do not use (disable) VAD ability of the vocoder under consideration.
VE_MEDIA_VAD_ENABLE_BOOLEAN_TRUE	Use (enable) VAD ability of the vocoder under consideration.

B.110 t_ve_media_id_e

The **t_ve_media_id_e** enumeration is part of the **t_ve_channel_packet_configuration_s** structure (defined in Appendix B.21.3 on Page 293) and the **t_ve_media_field_s** structure (defined in Appendix B.28 on Page 317). The **t_ve_media_id_e** enumeration determines the packet-side data format to be expected by the VeriCall Edge system from the registered data manager callback functions. The **t_ve_media_id_e** enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  NONE_MEDIA_ID              = 0,
  CN_MEDIA_ID                = 1,
  G711U_MEDIA_ID             = 2,
  G711A_MEDIA_ID             = 3,
  G729_AB_MEDIA_ID           = 4,
  TE_MEDIA_ID                = 5,
  GIPS_EG711_A_MEDIA_ID      = 6,
  GIPS_EG711_U_MEDIA_ID      = 7,
  MPEG4_SQCIF_MEDIA_ID       = 8,
  MPEG4_QCIF_MEDIA_ID        = 9,
  MPEG4_CIF_MEDIA_ID         = 10,
  H263_SQCIF_MEDIA_ID        = 11,
  H263_QCIF_MEDIA_ID         = 12,
  H263_CIF_MEDIA_ID          = 13,
  H263_1998_SQCIF_MEDIA_ID   = 14,
  H263_1998_QCIF_MEDIA_ID    = 15,
  H263_1998_CIF_MEDIA_ID     = 16,
  G723_1_MEDIA_ID            = 17,
  GSM_AMR_MEDIA_ID           = 18,
  G726_16_MEDIA_ID           = 19,
  G726_24_MEDIA_ID           = 20,
  G726_32_MEDIA_ID           = 21,
  G726_40_MEDIA_ID           = 22,
  ILBC_MEDIA_ID              = 23,
  MAX_MEDIA_ID               = 24
}t_ve_media_id_e;
```

The enumeration elements are as follows:

Element	Definition
NONE_MEDIA_ID	There is no media type defined.
CN_MEDIA_ID	The comfort noise (CN) media type.
G711U_MEDIA_ID	The G.711 u-Law media type.
G711A_MEDIA_ID	The G.711 a-law media type.
G729_AB_MEDIA_ID	The G.729AB media type.
TE_MEDIA_ID	The telephony event (TE) media type.
GIPS_EG711_A_MEDIA_ID	The GIPS enhanced G.711 a-law media type.
GIPS_EG711_U_MEDIA_ID	The GIPS enhanced G.711 u-law media type.
MPEG4_SQCIF_MEDIA_ID	The MPEG4 Sub-QCIF video media type.

Element	Definition
MPEG4_QCIF_MEDIA_ID	The MPEG4 Quarter-CIF (QCIF) video media type.
MPEG4_CIF_MEDIA_ID	The MPEG4 Common Intermediate Format (CIF) video media type.
H263_SQCIF_MEDIA_ID	The H.263 Sub-QCIF video media type.
H263_QCIF_MEDIA_ID	The H.263 Quarter-CIF (QCIF) video media type.
H263_CIF_MEDIA_ID	The H.263 Common Intermediate Format (CIF) video media type.
H263_1998_SQCIF_MEDIA_ID	The H.263 1998 Sub-QCIF video media type.
H263_1998_QCIF_MEDIA_ID	The H.263 1998 Quarter-CIF (QCIF) video media type.
H263_1998_CIF_MEDIA_ID	The H.263 1998 Common Intermediate Format (CIF) video media type.
G723_1_MEDIA_ID	The G.723_1 media type.
GSM_AMR_MEDIA_ID	The GSM-AMR media type. Reserved for future use.
G726_16_MEDIA_ID	The G.726 at 16kbits/s media type.
G726_24_MEDIA_ID	The G.726 at 24kbits/s media type.
G726_32_MEDIA_ID	The G.726 at 32kbits/s media type.
G726_40_MEDIA_ID	The G.726 at 40kbits/s media type.
ILBC_MEDIA_ID	The iLBC media type.
MAX_MEDIA_ID	The maximum number of media ID's available and should be the last entry of the enumerated list.

The Global IP Sound (GIPS) enhanced G.711 algorithm is not part of the default VeriCall Edge system configuration. Please contact Trinity Convergence if you require more information about including GIPS within the VeriCall Edge software.

B.111 t_ve_media_eg711_codec_and_law_e

The **t_ve_media_eg711_codec_and_law_e** enumeration is part of the GIPS enhanced G.711 encoder and decoder configuration structures defined in Appendix B.29.19 on Page 337 and Appendix B.29.20 on Page 338. The **t_ve_media_eg711_codec_and_law_e** enumeration determines the codec and law to be used within the GIPS enhanced G.711 encoder and decoder. The **t_ve_media_eg711_codec_and_law_e** enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_NETEQFIX_GIPS_IPCMU = 0,
  VE_MEDIA_NETEQFIX_GIPS_IPCMA = 1,
  VE_MEDIA_NETEQFIX_GIPS_G711U = 2,
  VE_MEDIA_NETEQFIX_GIPS_G711A = 3
}t_ve_media_eg711_codec_and_law_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_NETEQFIX_GIPS_IPCMU	Determine the codec and law as being GIPS IPC μ-law.
VE_MEDIA_NETEQFIX_GIPS_IPCMA	Determine the codec and law as being GIPS IPC a-law.
VE_MEDIA_NETEQFIX_GIPS_G711U	Determine the codec and law as being GIPS enhanced G.711 μ-law.
VE_MEDIA_NETEQFIX_GIPS_G711A	Determine the codec and law as being GIPS enhanced G.711 a-law.

The Global IP Sound (GIPS) enhanced G.711 algorithm is not part of the default VeriCall Edge system configuration. Please contact Trinity Convergence if you require more information about including GIPS within the VeriCall Edge software.

B.112 t_ve_media_video_frame_type_selection_e

The **t_ve_media_video_frame_type_selection_e** enumeration is part of the **t_ve_media_video_frame_type_selection_s** structure, described in Appendix B.29.18 on Page 336. The **t_ve_media_video_frame_type_selection_e** enumeration determines the video framing type to be used for the video encoder. The enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_VIDEO_FRAME_TYPE_SELECTION_I_FRAMES_ONLY = 0,
  VE_MEDIA_VIDEO_FRAME_TYPE_SELECTION_I_FRAMES_AND_P_FRAMES,
  VE_MEDIA_VIDEO_FRAME_TYPE_SELECTION_P_FRAMES_ONLY,
}t_ve_media_video_frame_type_selection_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_VIDEO_FRAME_TYPE_ SELECTION_I_FRAMES_ONLY	The framing type for the current image is “I-frames” only.
VE_MEDIA_VIDEO_FRAME_TYPE_ SELECTION_I_FRAMES_AND_P_FRAMES	The framing type for the current image is “I-frames” and “P-frames”.
VE_MEDIA_VIDEO_FRAME_TYPE_ SELECTION_P_FRAMES_ONLY	The framing type for the current image is “P-frames” only.

B.113 t_ve_media_video_encoding_mode_e

The **t_ve_media_video_encoding_mode_e** enumeration is part of the **t_ve_media_video_encoder_h263_algorithm_configuration_s** structure, described in Appendix B.29.14 on Page 325, and the **t_ve_media_video_encoder_mpeg4_algorithm_configuration_s** structure, described in Appendix B.29.16 on Page 327. The **t_ve_media_video_encoding_mode_e** enumeration determines the video mode of operation for this video encoder. The enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_VIDEO_ENCODING_MODE_SQCIF = 0,
  VE_MEDIA_VIDEO_ENCODING_MODE_QCIF,
  VE_MEDIA_VIDEO_ENCODING_MODE_CIF
}t_ve_media_video_encoding_mode_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_VIDEO_ENCODING_MODE_SQCIF	The video encoding format to be used is Sub-QCIF.
VE_MEDIA_VIDEO_ENCODING_MODE_QCIF	The video encoding format to be used is Quarter-CIF (QCIF).
VE_MEDIA_VIDEO_ENCODING_MODE_CIF	The video encoding format to be used is Common Intermediate Format (CIF).

B.114 t_ve_media_video_profile_selection_e

The **t_ve_media_video_profile_selection_e** enumeration is part of the **t_ve_media_video_encoder_h263_algorithm_configuration_s** structure defined in Appendix B.29.14 on Page 325. The **t_ve_media_video_profile_selection_e** enumeration determines the profile to be used for this video encoder. The enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_VIDEO_PROFILE_P0_SELECTION = 0,
  VE_MEDIA_VIDEO_PROFILE_P3_SELECTION
}t_ve_media_video_profile_selection_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_VIDEO_H263_PROFILE_P0_SELECTION	The encoding video profile to be used is P0.
VE_MEDIA_VIDEO_H263_PROFILE_P3_SELECTION	Reserved for future use.

This user application should not attempt to use the H.263 P3 Profile. This element is reserved for future use.

B.115 t_ve_media_video_encoding_mode_rate_e

The **t_ve_media_video_encoding_mode_rate_e** enumeration is part of the **t_ve_media_video_encoding_mode_rate_s** structure defined in Appendix B.29.23 on Page 341. The **t_ve_media_video_encoding_mode_rate_e** enumeration determines the H.263 annex profile to be used for this video encoder. The enumeration is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_MEDIA_VIDEO_ENCODING_MODE_RATE_IMMEDIATE,
  VE_MEDIA_VIDEO_ENCODING_MODE_RATE_CONFIGURED
}t_ve_media_video_encoding_mode_rate_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_VIDEO_ENCODING_MODE_RATE_IMMEDIATE	The encoding H.263 rate is immediate in that this element forces an immediate I-frame and then resumes the previous I-Frame rate.
VE_MEDIA_VIDEO_ENCODING_MODE_RATE_CONFIGURED	The encoding H.263 rate is configured.

B.116 t_ve_media_video_quantization_param_e

The **t_ve_media_video_quantization_param_e** enumeration is part of the following structures:

- t_ve_media_video_encoder_mpeg4_algorithm_configuration_s described in Appendix B.29.16 on Page 327.
- t_ve_media_video_pid_rate_control_s described in Appendix B.29.16.5 on Page 332.
- t_ve_media_mpeg4_configuration_s described in Appendix B.28.8 on Page 314.

The **t_ve_media_video_quantization_param_e** enumeration determines the MPEG-4 quantization parameter to be used. The enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_INVALID = 0,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_1,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_2,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_3,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_4,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_5,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_6,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_7,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_8,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_9,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_10,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_11,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_12,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_13,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_14,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_15,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_16,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_17,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_18,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_19,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_20,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_21,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_22,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_23,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_24,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_25,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_26,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_27,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_28,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_29,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_30,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_31,
  VE_MEDIA_VIDEO_QUANTIZATION_PARAM_COUNT,
}t_ve_media_video_quantization_param_e;
```

Continued >>

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_INVALID	This element is used internally by the VeriCall Edge system and should not be used by the user application.
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_1	The integer value to be used as the default Quantization Parameter (QP) for the MPEG-4 encoding.
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_2	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_3	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_4	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_5	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_6	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_7	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_8	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_9	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_10	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_11	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_12	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_13	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_14	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_15	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_16	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_17	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_18	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_19	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_20	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_21	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_22	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_23	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_24	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_25	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_26	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_27	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_28	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_29	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_30	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_31	
VE_MEDIA_VIDEO_QUANTIZATION_PARAM_COUNT	This element is used internally by the VeriCall Edge system and should not be used by the user application.

B.117 t_ve_media_video_config_state_e

The **t_ve_media_video_config_state_e** enumeration is part of the following structures:

- **t_ve_media_video_encoder_mpeg4_algorithm_configuration_s** described in Appendix B.29.16 on Page 327.
- **t_ve_media_video_user_data_s** described in Appendix B.29.16.1 on Page 328.
- **t_ve_media_video_stream_type_info_s** described in Appendix B.29.16.3 on Page 330.
- **t_ve_media_video_cyclic_intra_refresh_s** described in Appendix B.29.16.4 on Page 331.
- **t_ve_media_video_pid_rate_control_s** described in Appendix B.29.16.5 on Page 332.
- **t_ve_media_video_compression_params_s** described in Appendix B.29.16.8 on Page 335.

The **t_ve_media_video_config_state_e** enumeration determines the state of the MPEG-4 video channel configuration. The enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_VIDEO_CONFIG_INVALID = 0,
  VE_MEDIA_VIDEO_CONFIG_DISABLE,
  VE_MEDIA_VIDEO_CONFIG_ENABLE,
  VE_MEDIA_VIDEO_CONFIG_COUNT
}t_ve_media_video_config_state_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_VIDEO_CONFIG_INVALID	This element is used internally by the VeriCall Edge system and should not be used by the user application.
VE_MEDIA_VIDEO_CONFIG_DISABLE	Disable the video channel configuration.
VE_MEDIA_VIDEO_CONFIG_ENABLE	Enable the video channel configuration.
VE_MEDIA_VIDEO_CONFIG_COUNT	This element is used internally by the VeriCall Edge system and should not be used by the user application.

B.118 t_ve_media_video_simple_profile_e

The **t_ve_media_video_simple_profile_e** enumeration is part of the **t_ve_media_mpeg4_configuration_s** structure, described in Appendix B.28.8 on Page 314, and the **t_ve_media_video_stream_type_info_s** structure, described in Appendix B.29.16.3 on Page 330. The **t_ve_media_video_simple_profile_e** enumeration determines the simple profile to be used with the MPEG-4 video channel configuration. The enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_VIDEO_SIMPLE_PROFILE_LEVEL_INVALID = 0,
  VE_MEDIA_VIDEO_SIMPLE_PROFILE_LEVEL_1 = 1,
  VE_MEDIA_VIDEO_SIMPLE_PROFILE_LEVEL_2 = 2,
  VE_MEDIA_VIDEO_SIMPLE_PROFILE_LEVEL_3 = 3,
  VE_MEDIA_VIDEO_SIMPLE_PROFILE_LEVEL_0 = 8,
  VE_MEDIA_VIDEO_SIMPLE_PROFILE_LEVEL_COUNT
}t_ve_media_video_simple_profile_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_VIDEO_CONFIG_INVALID	This element is used internally by the VeriCall Edge system and should not be used by the user application.
VE_MEDIA_VIDEO_SIMPLE_PROFILE_LEVEL_1	The simple profile to be used. See the MPEG-4 standard for information on these simple profiles. These profiles set such things as the upper limits for the maximum bit-rate, frame rate and frame size.
VE_MEDIA_VIDEO_SIMPLE_PROFILE_LEVEL_2	
VE_MEDIA_VIDEO_SIMPLE_PROFILE_LEVEL_3	
VE_MEDIA_VIDEO_SIMPLE_PROFILE_LEVEL_0	
VE_MEDIA_VIDEO_CONFIG_COUNT	This element is used internally by the VeriCall Edge system and should not be used by the user application.

B.119 t_ve_media_video_profile_level_e

The **t_ve_media_video_profile_level_e** enumeration is part of the **t_ve_media_video_encoder_h263_algorithm_configuration_s** structure defined in Appendix B.29.14 on Page 325, and the **t_ve_media_h263_configuration_s** structure, described in Appendix B.28.7 on Page 313. The **t_ve_media_video_profile_level_e** enumeration determines the profile level to be used with the H.263 video channel configuration. The enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_VIDEO_PROFILE_LEVEL_INVALID = 0,
  VE_MEDIA_VIDEO_PROFILE_LEVEL_0,
  VE_MEDIA_VIDEO_PROFILE_LEVEL_1,
  VE_MEDIA_VIDEO_PROFILE_LEVEL_2,
  VE_MEDIA_VIDEO_PROFILE_LEVEL_3,
  VE_MEDIA_VIDEO_PROFILE_LEVEL_4,
  VE_MEDIA_VIDEO_PROFILE_LEVEL_5,
  VE_MEDIA_VIDEO_PROFILE_LEVEL_6,
  VE_MEDIA_VIDEO_PROFILE_LEVEL_7,
  VE_MEDIA_VIDEO_PROFILE_LEVEL_8,
  VE_MEDIA_VIDEO_PROFILE_LEVEL_9,
  VE_MEDIA_VIDEO_PROFILE_LEVEL_10,
  VE_MEDIA_VIDEO_PROFILE_LEVEL_COUNT
}t_ve_media_video_profile_level_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_VIDEO_PROFILE_LEVEL_INVALID	This element is used internally by the VeriCall Edge system and should not be used by the user application.
VE_MEDIA_VIDEO_PROFILE_LEVEL_0	The profile level to be used with the H.263 video channel configuration.
VE_MEDIA_VIDEO_PROFILE_LEVEL_1	
VE_MEDIA_VIDEO_PROFILE_LEVEL_2	
VE_MEDIA_VIDEO_PROFILE_LEVEL_3	
VE_MEDIA_VIDEO_PROFILE_LEVEL_4	
VE_MEDIA_VIDEO_PROFILE_LEVEL_5	
VE_MEDIA_VIDEO_PROFILE_LEVEL_6	
VE_MEDIA_VIDEO_PROFILE_LEVEL_7	
VE_MEDIA_VIDEO_PROFILE_LEVEL_8	
VE_MEDIA_VIDEO_PROFILE_LEVEL_9	
VE_MEDIA_VIDEO_PROFILE_LEVEL_10	
VE_MEDIA_VIDEO_PROFILE_LEVEL_COUNT	This element is used internally by the VeriCall Edge system and should not be used by the user application.

B.120 t_ve_media_video_stream_type_e

The **t_ve_media_video_stream_type_e** enumeration is part of the **t_ve_media_video_encoder_mpeg4_algorithm_configuration_s** described in Appendix B.29.16 on Page 327. The **t_ve_media_video_stream_type_e** enumeration determines the MPEG-4 video stream type to be used in this video channel configuration. The enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_VIDEO_STREAM_TYPE_PLAIN = 1,
  VE_MEDIA_VIDEO_STREAM_TYPE_VP
}t_ve_media_video_stream_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_VIDEO_STREAM_TYPE_PLAIN	The MPEG-4 stream type is plain.
VE_MEDIA_VIDEO_STREAM_TYPE_VP	The MPEG-4 stream type is video packets.

B.121 t_ve_media_vad_state_e

The **t_ve_media_vad_state_e** enumeration is part of the media configuration structures for **t_ve_media_g729a_configuration_s**, described in Appendix B.28.2 on Page 308, and **t_ve_media_g723_1_configuration_s,** described in Appendix B.28.3 on Page 309. The **t_ve_media_vad_state_e** enumeration determines whether VAD is enabled or disabled within the configuration. The enumeration is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_MEDIA_VAD_STATE_DISABLE,
  VE_MEDIA_VAD_STATE_ENABLE,
}t_ve_media_vad_state_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_VAD_STATE_DISABLE	Disable VAD functionality.
VE_MEDIA_VAD_STATE_ENABLE	Enable VAD functionality.

B.122 t_ve_media_g723_1_rate_e

The **t_ve_media_g723_1_rate_e** enumeration is used within the **t_ve_media_g723_1_configuration_s,** described in Appendix B.28.3 on Page 309. The **t_ve_media_g723_1_rate_e** enumeration determines the data rate for this configuration. The enumeration is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_MEDIA_G723_1_RATE_5300BPS,
  VE_MEDIA_G723_1_RATE_6300BPS,
}t_ve_media_g723_1_rate_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_G723_1_RATE_5300BPS	The data rate for this configuration is 5.3Kbps.
VE_MEDIA_G723_1_RATE_6300BPS	The data rate for this configuration is 6.3Kbps.

B.123 t_ve_media_g723_1_mode_e

The **t_ve_media_g723_1_mode_e** enumeration is used within the **t_ve_media_g723_1_encoder_algorithm_configuration_s** structure as defined in Appendix B.29.5 on Page 319. The **t_ve_media_g723_1_mode_e** enumeration determines the data rate for this encoder configuration. The enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_G723_1_MODE_5300,
  VE_MEDIA_G723_1_MODE_6300,
  VE_MEDIA_G723_1_MODE_UNCHANGED
}t_ve_media_g723_1_mode_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_G723_1_MODE_5300	The data rate for this configuration is 5.3Kbps.
VE_MEDIA_G723_1_MODE_6300	The data rate for this configuration is 6.3Kbps.
VE_MEDIA_G723_1_MODE_UNCHANGED	The data rate for this configuration is unchanged.

B.124 t_ve_media_g723_1_hpf_enable_boolean_e

The **t_ve_media_g723_1_hpf_enable_boolean_e** enumeration is used within the **t_ve_media_g723_1_encoder_algorithm_configuration_s** structure as defined in Appendix B.29.5 on Page 319. The **t_ve_media_g723_1_hpf_enable_boolean_e** enumeration determines the state of the encoder high pass filter for this configuration. The enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_G723_1_HPF_ENABLE_BOOLEAN_FALSE,
  VE_MEDIA_G723_1_HPF_ENABLE_BOOLEAN_TRUE
}t_ve_media_g723_1_hpf_enable_boolean_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_G723_1_HPF_ENABLE_BOOLEAN_FALSE	Disable the high pass filter.
VE_MEDIA_G723_1_HPF_ENABLE_BOOLEAN_TRUE	Enable the high pass filter.

B.125 t_ve_media_g723_1_pf_enable_boolean_e

The **t_ve_media_g723_1_pf_enable_boolean_e** enumeration is used within the **t_ve_media_g723_1_decoder_algorithm_configuration_s** structure as defined in Appendix B.29.6 on Page 319. The **t_ve_media_g723_1_pf_enable_boolean_e** enumeration determines whether the post filter is to be used for this decoder configuration. The enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_G723_1_PF_ENABLE_BOOLEAN_FALSE,
  VE_MEDIA_G723_1_PF_ENABLE_BOOLEAN_TRUE
}t_ve_media_g723_1_pf_enable_boolean_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_G723_1_PF_ENABLE_BOOLEAN_FALSE	Disable the post filter.
VE_MEDIA_G723_1_PF_ENABLE_BOOLEAN_TRUE	Enable the post filter.

B.126 t_ve_media_gsm_amr_rate_e

The **t_ve_media_gsm_amr_rate_e** enumeration is used within the **t_ve_media_gsm_amr_configuration_s** as defined in Appendix B.28.5 on Page 310. The **t_ve_media_gsm_amr_rate_e** enumeration determines the data rate for this configuration. The enumeration is defined in **ve_defs.h** and is described below:

```
Typedef enum{
  VE_MEDIA_GSM_AMR_RATE_4750BPS
  VE_MEDIA_GSM_AMR_RATE_5150BPS,
  VE_MEDIA_GSM_AMR_RATE_5900BPS,
  VE_MEDIA_GSM_AMR_RATE_6700BPS,
  VE_MEDIA_GSM_AMR_RATE_7400BPS,
  VE_MEDIA_GSM_AMR_RATE_7950BPS,
  VE_MEDIA_GSM_AMR_RATE_10200BPS,
  VE_MEDIA_GSM_AMR_RATE_12200BPS,
}t_ve_media_gsm_amr_rate_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_GSM_AMR_RATE_4750BPS	The data rate for this configuration is 4.75Kbps.
VE_MEDIA_GSM_AMR_RATE_5150BPS	The data rate for this configuration is 5.15Kbps.
VE_MEDIA_GSM_AMR_RATE_5900BPS	The data rate for this configuration is 5.90Kbps.
VE_MEDIA_GSM_AMR_RATE_6700BPS	The data rate for this configuration is 6.70Kbps.
VE_MEDIA_GSM_AMR_RATE_7400BPS	The data rate for this configuration is 7.40Kbps.
VE_MEDIA_GSM_AMR_RATE_7950BPS	The data rate for this configuration is 7.95Kbps.
VE_MEDIA_GSM_AMR_RATE_10200BPS	The data rate for this configuration is 10.20Kbps.
VE_MEDIA_GSM_AMR_RATE_12200BPS	The data rate for this configuration is 12.20Kbps.

B.127 t_ve_media_gsm_amr_mode_e

The **t_ve_media_gsm_amr_mode_e** enumeration is used within the **t_ve_media_gsm_amr_encoder_algorithm_configuration_s** structure as defined in Appendix B.29.9 on Page 321. The **t_ve_media_gsm_amr_mode_e** enumeration determines the mode of operation for this encoder configuration. The enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_GSM_AMR_ENCODE_MODE_475 = 0,
  VE_MEDIA_GSM_AMR_ENCODE_MODE_515,
  VE_MEDIA_GSM_AMR_ENCODE_MODE_59,
  VE_MEDIA_GSM_AMR_ENCODE_MODE_67,
  VE_MEDIA_GSM_AMR_ENCODE_MODE_74,
  VE_MEDIA_GSM_AMR_ENCODE_MODE_795,
  VE_MEDIA_GSM_AMR_ENCODE_MODE_102,
  VE_MEDIA_GSM_AMR_ENCODE_MODE_122,
  VE_MEDIA_GSM_AMR_ENCODE_MODE_UNCHANGED
}t_ve_media_gsm_amr_mode_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_GSM_AMR_ENCODE_MODE_475	The data rate for this configuration is 4.75Kbps.
VE_MEDIA_GSM_AMR_ENCODE_MODE_515	The data rate for this configuration is 5.15Kbps.
VE_MEDIA_GSM_AMR_ENCODE_MODE_59	The data rate for this configuration is 5.90Kbps.
VE_MEDIA_GSM_AMR_ENCODE_MODE_67	The data rate for this configuration is 6.70Kbps.
VE_MEDIA_GSM_AMR_ENCODE_MODE_74	The data rate for this configuration is 7.40Kbps.
VE_MEDIA_GSM_AMR_ENCODE_MODE_795	The data rate for this configuration is 7.95Kbps.
VE_MEDIA_GSM_AMR_ENCODE_MODE_102	The data rate for this configuration is 10.20Kbps.
VE_MEDIA_GSM_AMR_ENCODE_MODE_122	The data rate for this configuration is 12.20Kbps.
VE_MEDIA_GSM_AMR_ENCODE_MODE_UNCHANGED	The data rate for this configuration is unchanged.

B.128 t_ve_media_gsm_amr_vad_mode_e

The **t_ve_media_gsm_amr_vad_mode_e** enumeration is used within the **t_ve_media_gsm_amr_encoder_algorithm_configuration_s** structure as defined in Appendix B.29.9 on Page 321. The **t_ve_media_gsm_amr_vad_mode_e** enumeration determines the discontinuous transmission mode to be used for this encoder configuration. The enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_GSM_AMR_DTX_MODE_OFF,
  VE_MEDIA_GSM_AMR_DTX_MODE_ON_VAD1,
  VE_MEDIA_GSM_AMR_DTX_MODE_ON_VAD2,
  VE_MEDIA_GSM_AMR_DTX_MODE_UNCHANGED
}t_ve_media_gsm_amr_vad_mode_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_GSM_AMR_DTX_MODE_OFF	Do not use discontinuous transmission.
VE_MEDIA_GSM_AMR_DTX_MODE_ON_VAD1	Use discontinuous transmission VAD algorithm 1.
VE_MEDIA_GSM_AMR_DTX_MODE_ON_VAD2	Use discontinuous transmission VAD algorithm 2.
VE_MEDIA_GSM_AMR_DTX_MODE_UNCHANGED	Do not update the discontinuous transmission mode for this configuration.

B.129 t_ve_media_gsm_amr_vad_state_e

The **t_ve_media_gsm_amr_vad_state_e** enumeration is used within the **t_ve_media_gsm_amr_encoder_algorithm_configuration_s** structure as defined in Appendix B.29.9 on Page 321. The **t_ve_media_gsm_amr_vad_state_e** enumeration determines the whether the GSM-AMR VAD should be enabled and which type of VAD should be used. The **t_ve_media_gsm_amr_vad_state_e** enumerated type is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_MEDIA_GSM_AMR_VAD_STATE_DISABLE,
  VE_MEDIA_GSM_AMR_VAD1_STATE_ENABLE,
  VE_MEDIA_GSM_AMR_VAD2_STATE_ENABLE,
}t_ve_media_gsm_amr_vad_state_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_GSM_AMR_VAD_STATE_DISABLE	Disable the GSM-AMR VAD.
VE_MEDIA_GSM_AMR_VAD1_STATE_ENABLE	Enable the lightweight GSM-AMR VAD.
VE_MEDIA_GSM_AMR_VAD2_STATE_ENABLE	Enable the GSM-AMR VAD.

B.130 t_ve_media_stream_direction_e

The **t_ve_media_stream_direction_e** enumeration is used within the **t_ve_media_description_s** structure as defined in Appendix B.27 on Page 302. The **t_ve_media_stream_direction_e** enumeration determines the media stream direction to be used. The enumeration is defined in **ve_media.h** and is described below:

```
typedef enum{
  VE_MEDIA_STREAM_DIRECTION_SENDRECV = 0,
  VE_MEDIA_STREAM_DIRECTION_RECVONLY,
  VE_MEDIA_STREAM_DIRECTION_SENDONLY,
  VE_MEDIA_STREAM_DIRECTION_INACTIVE,
}t_ve_media_stream_direction_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MEDIA_STREAM_DIRECTION_SENDRECV	Both send and receive streams active.
VE_MEDIA_STREAM_DIRECTION_RECVONLY	The receive stream is the only active stream.
VE_MEDIA_STREAM_DIRECTION_SENDONLY	The send stream is the only active stream.
VE_MEDIA_STREAM_DIRECTION_INACTIVE	Both directions for the stream are disabled.

B.131 t_ve_last_stream_state_e

The **t_ve_last_stream_state_e** enumeration is used within the **t_ve_last_stream_configuration_s** structure as defined in Appendix B.5 on Page 257. The **t_ve_last_stream_state_e** enumeration determines the media stream processing state. The enumeration is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_LAST_STREAM_STATE_ILLEGAL,
  VE_LAST_STREAM_STATE_AWAITING_PROCESS,
  VE_LAST_STREAM_STATE_ACTIVE,
  VE_LAST_STREAM_STATE_INACTIVE,
  VE_LAST_STREAM_STATE_HOLD,
  VE_LAST_STREAM_STATE_COUNT
}t_ve_last_stream_state_e;
```

The enumeration elements are as follows:

Element	Definition
VE_LAST_STREAM_STATE_ILLEGAL	VeriCall Edge never received a start stream message or an invalid start stream configuration has been received.
VE_LAST_STREAM_STATE_AWAITING_PROCESS	The stream configuration is queued and awaiting processing by VeriCall Edge.
VE_LAST_STREAM_STATE_ACTIVE	The start stream configuration is currently active.
VE_LAST_STREAM_STATE_INACTIVE	A stop stream commend has been received and the stream is currently not in use.
VE_LAST_STREAM_STATE_HOLD	The stream is currently on hold.
VE_LAST_STREAM_STATE_COUNT	Used internally by VeriCall Edge and should not be used by the user application.

B.132 t_ve_module_state_e

The **t_ve_module_state_e** enumeration is used within the **t_ve_audio_video_module_state_s** structure, described in Appendix B.6.1 on Page 259, and the **t_ve_audio_module_state_s** structure, as described in Appendix B.6.2 on page 259. The **t_ve_module_state_e** enumeration determines the current state of the specified VeriCall Edge module. The enumeration is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_MODULE_STATE_ILLEGAL,
  VE_MODULE_STATE_IDLE,
  VE_MODULE_STATE_ACTIVE,
  VE_MODULE_STATE_COUNT,
}t_ve_module_state_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MODULE_STATE_ILLEGAL	Used internally by VeriCall Edge and should not be used by the user application.
VE_MODULE_STATE_IDLE	The current module state is idle.
VE_MODULE_STATE_ACTIVE	The current module state is active.
VE_MODULE_STATE_COUNT	Used internally by VeriCall Edge and should not be used by the user application.

B.133 t_ve_module_class_e

The **t_ve_module_class_e** enumeration is used within the **t_ve_run_vericall_edge_config_s** structure as defined in Appendix B.6 on Page 258. The **t_ve_module_class_e** enumeration determines the current VeriCall Edge module class type. The enumeration is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_MODULE_CLASS_AUDIO_VIDEO = 0,
  VE_MODULE_CLASS_AUDIO_ONLY
}t_ve_module_class_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MODULE_CLASS_AUDIO_VIDEO	The VeriCall Edge module class is applicable for audio and video.
VE_MODULE_CLASS_AUDIO_ONLY	The VeriCall Edge module class is applicable for audio only.

B.134 t_ve_network_control_type_e

The **t_ve_network_control_type_e** enumeration is used by **VE_ControlNetwork**. The **t_ve_network_control_type_e** enumeration determines the network control element of the **t_ve_network_control_u** union, defined in Appendix B.30 on Page 342, to be modified. The enumeration is defined in **ve_control_network.h** and is described below:

```
typedef enum{
  VE_NETWORK_CONTROL_TYPE_INVALID_SELECTION = 0,
  VE_NETWORK_CONTROL_TYPE_TOS,
  VE_NETWORK_CONTROL_TYPE_NUM_SELECTIONS
}t_ve_network_control_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_NETWORK_CONTROL_TYPE_INVALID_SELECTION	This is an invalid control type selection.
VE_NETWORK_CONTROL_TYPE_TOS	Modify the default Type of Service (TOS) settings of the network.
VE_NETWORK_CONTROL_TYPE_NUM_SELECTIONS	The number of control type elements in this enumeration.

B.135 t_ve_network_connection_type_e

The **t_ve_network_connection_type_e** enumeration is used within the **t_ve_network_connection_s** structure described in Appendix B.64 on Page 498. The **t_ve_network_connection_type_e** enumeration determines the network connection type to be used or that was used. The enumeration is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_NETWORK_CONNECTION_TYPE_INVALID=0,
  VE_NETWORK_CONNECTION_TYPE_SIP_REGISTRATION,
  VE_NETWORK_CONNECTION_TYPE_SIP_DIALOGUE,
  VE_NETWORK_CONNECTION_TYPE_MEDIA_RTP,
  VE_NETWORK_CONNECTION_TYPE_MEDIA_RTCP,
  VE_NUM_NETWORK_CONNECTION_TYPES
}t_ve_network_connection_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_NETWORK_CONNECTION_TYPE_INVALID	This element is used internally by VeriCall Edge and should not be used by the user application.
VE_NETWORK_CONNECTION_TYPE_SIP_REGISTRATION	The connection type is a SIP registration network connection.
VE_NETWORK_CONNECTION_TYPE_SIP_DIALOGUE	The connection type is a SIP dialog network connection
VE_NETWORK_CONNECTION_TYPE_MEDIA_RTP	The connection type is a RTP network connection from the VeriCall Edge Media Engine.
VE_NETWORK_CONNECTION_TYPE_MEDIA_RTCP	The connection type is a RTCP network connection from the VeriCall Edge Media Engine.
VE_NUM_NETWORK_CONNECTION_TYPES	This element is used internally by VeriCall Edge and should not be used by the user application.

B.136 t_ve_network_connection_event_e

The **t_ve_network_connection_event_e** enumeration is used within the **t_ve_user_msg_network_connection_event_s** structure as described in Appendix B.33.64 on Page 417. The **t_ve_network_connection_event_e** enumeration determines the network connection event that has occurred. The enumeration is defined in **ve_events.h** and is described below:

```
typedef enum{
  VE_NETWORK_CONNECTION_EVENT_INVALID=0,
  VE_NETWORK_CONNECTION_EVENT_OPENED,
  VE_NETWORK_CONNECTION_EVENT_CLOSED,
  VE_NUM_NETWORK_CONNECTION_EVENTS
}t_ve_network_connection_event_e;
```

The enumeration elements are as follows:

Element	Definition
VE_NETWORK_CONNECTION_EVENT_INVALID	This element is used internally by VeriCall Edge and should not be used by the user application.
VE_NETWORK_CONNECTION_EVENT_OPENED	A network connection was opened.
VE_NETWORK_CONNECTION_EVENT_CLOSED	A network connection was closed.
VE_NUM_NETWORK_CONNECTION_EVENTS	This element is used internally by VeriCall Edge and should not be used by the user application.

B.137 t_ve_net_lookup_type_e

The **t_ve_net_lookup_type_e** enumeration is used within the **t_ve_net_lookup_query_s** structure as described in Appendix B.66 on Page 500. The **t_ve_net_lookup_type_e** enumeration determines the lookup type used in the network lookup request. The enumeration is defined in **ve_net_defs.h** and is described below:

```
typedef enum{
  VE_NET_LOOKUP_TYPE_INVALID,
  VE_NET_LOOKUP_TYPE_IPV4_ADDRESS,
  VE_NET_LOOKUP_TYPE_SERVER,
  VE_NET_LOOKUP_TYPE_NAPTR,
  VE_NET_LOOKUP_TYPE_NAPTR_ENUM
}t_ve_net_lookup_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_NET_LOOKUP_TYPE_INVALID	This element is used internally by VeriCall Edge and should not be used by the user application.
VE_NET_LOOKUP_TYPE_IPV4_ADDRESS	The lookup type requested is an IPv4 address.
VE_NET_LOOKUP_TYPE_SERVER	The lookup type requested is a server address.
VE_NET_LOOKUP_TYPE_NAPTR	The lookup type requested is a NAPTR address.
VE_NET_LOOKUP_TYPE_NAPTR_ENUM	The lookup type requested is a NAPTR ENUM address.

B.138 t_ve_net_service_type_e

The **t_ve_net_service_type_e** enumeration is used within the following structures:

- **t_ve_net_lookup_server_s** as described in Appendix B.68 on Page 501.
- **t_ve_net_lookup_naptr_s** as described in Appendix B.69 on Page 502.
- **t_ve_net_lookup_naptr_enum_s** as described in Appendix B.70 on Page 503.

The **t_ve_net_service_type_e** enumeration determines the service type used in the network lookup request. The enumeration is defined in **ve_net_defs.h** and is described below:

```
typedef enum{
  VE_NET_SERVICE_TYPE_INVALID,
  VE_NET_SERVICE_TYPE_SIP,
  VE_NET_SERVICE_TYPE_STUN
}t_ve_net_service_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_NET_SERVICE_TYPE_INVALID	This element is used internally by VeriCall Edge and should not be used by the user application.
VE_NET_SERVICE_TYPE_SIP	The lookup service type is of type SIP.
VE_NET_SERVICE_TYPE_STUN	The lookup service type is of type STUN.

B.139 t_ve_net_service_protocol_e

The **t_ve_net_service_protocol_e** enumeration is used within the **t_ve_net_lookup_server_s** structure, described in Appendix B.68 on Page 501, and the **t_ve_net_lookup_naptr_s** structure, described in Appendix B.69 on Page 502. The **t_ve_net_service_protocol_e** enumeration determines the service protocol used in the network lookup request. The enumeration is defined in **ve_net_defs.h** and is described below:

```
typedef enum{
  VE_NET_SERVICE_PROTOCOL_INVALID,
  VE_NET_SERVICE_PROTOCOL_UDP,
  VE_NET_SERVICE_PROTOCOL_TCP
}t_ve_net_service_protocol_e;
```

The enumeration elements are as follows:

Element	Definition
VE_NET_SERVICE_PROTOCOL_INVALID	This element is used internally by VeriCall Edge and should not be used by the user application.
VE_NET_SERVICE_PROTOCOL_UDP	The lookup service protocol type is UDP.
VE_NET_SERVICE_PROTOCOL_TCP	The lookup service protocol type is TCP.

B.140 t_ve_net_lookup_record_type_e

The **t_ve_net_lookup_record_type_e** enumeration is used within the **t_ve_net_lookup_record_s** structure as described in Appendix B.33.65.1 on Page 419. The **t_ve_net_lookup_record_type_e** enumeration determines the network lookup record type requested. The enumeration is defined in **ve_net_defs.h** and is described below:

```
typedef enum{
  VE_NET_LOOKUP_RECORD_TYPE_INVALID,
  VE_NET_LOOKUP_RECORD_TYPE_IPV4_ADDRESS,
  VE_NET_LOOKUP_RECORD_TYPE_SERVER,
  VE_NET_LOOKUP_RECORD_TYPE_NAPTR
}t_ve_net_lookup_record_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_NET_LOOKUP_RECORD_TYPE_INVALID	This element is used internally by VeriCall Edge and should not be used by the user application.
VE_NET_LOOKUP_RECORD_TYPE_IPV4_ADDRESS	The lookup service record type is an IPv4 address.
VE_NET_LOOKUP_RECORD_TYPE_SERVER	The lookup service record type is a server.
VE_NET_LOOKUP_RECORD_TYPE_NAPTR	The lookup service record type is a NAPTR record.

B.141 t_ve_net_request_type_e

The **t_ve_net_request_type_e** enumeration is used within the

t_ve_user_msg_net_request_failed_s structure as described in Appendix 0 on Page 421. The **t_ve_net_request_type_e** enumeration determines the network request type that generated the failure event. The enumeration is defined in **ve_net_defs.h** and is described below:

```
typedef enum{
  VE_NET_REQUEST_TYPE_INVALID=0,
  VE_NET_REQUEST_TYPE_VE_NETLOOKUP,
  VE_NET_NUM_REQUEST_TYPES
}t_ve_net_request_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_NET_REQUEST_TYPE_INVALID	This element is used internally by VeriCall Edge and should not be used by the user application.
VE_NET_REQUEST_TYPE_VE_NETLOOKUP	The request type was a network lookup request.
VE_NET_NUM_REQUEST_TYPES	This element is used internally by VeriCall Edge and should not be used by the user application.

B.142 t_ve_net_request_failed_reason_e

The **t_ve_net_request_failed_reason_e** enumeration is used within the

t_ve_user_msg_net_request_failed_s structure as described in Appendix 0 on Page 421. The **t_ve_net_request_failed_reason_e** enumeration determines the reason the network request failed event was generated. The enumeration is defined in **ve_net_defs.h** and is described below:

```
typedef enum{
  VE_NET_REQUEST_FAILED_REASON_INVALID=0,
  VE_NET_REQUEST_FAILED_NO_FREE_JOB,
  VE_NET_REQUEST_FAILED_TRANSPORT_ERROR,
  VE_NET_REQUEST_FAILED_INVALID_PARAMETER,
  VE_NET_REQUEST_FAILED_DNS_ERROR,
  VE_NET_REQUEST_FAILED_INTERNAL_ERROR,
  VE_NET_NUM_REQUEST_FAILED_REASONS
}t_ve_net_request_failed_reason_e;
```

The enumeration elements are as follows:

Element	Definition
VE_NET_REQUEST_FAILED_REASON_INVALID	This element is used internally by VeriCall Edge and should not be used by the user application.
VE_NET_REQUEST_FAILED_NO_FREE_JOB	The request cannot be performed as there were no free jobs available.
VE_NET_REQUEST_FAILED_TRANSPORT_ERROR	The request failed due to a transport (network) failure.
VE_NET_REQUEST_FAILED_INVALID_PARAMETER	The request failed due to one or more invalid parameters.
VE_NET_REQUEST_FAILED_DNS_ERROR	The request failed due to a DNS messaging error or no DNS server address specified.
VE_NET_REQUEST_FAILED_INTERNAL_ERROR	The request failed due to an unspecified internal module error.
VE_NET_NUM_REQUEST_FAILED_REASONS	This element is used internally by VeriCall Edge and should not be used by the user application.

B.143 t_ve_audio_control_type_e

The **t_ve_audio_control_type_e** enumeration is used by **VE_ControlAudio** function calls to allow a number of user defined audio options to be enabled. This enumeration is defined in **ve_control_audio.h** and is described below.

```
typedef enum{
  VE_AUDIO_CONTROL_TYPE_RECORD_START = 0,
  VE_AUDIO_CONTROL_TYPE_RECORD_STOP,
  VE_AUDIO_CONTROL_TYPE_PLAY_ONCE_START,
  VE_AUDIO_CONTROL_TYPE_PLAY_LOOP_START,
  VE_AUDIO_CONTROL_TYPE_PLAY_STOP,
  VE_AUDIO_CONTROL_TYPE_AUDIO_LIP_SYNC_ADJUSTMENT,
  VE_AUDIO_CONTROL_TYPE_MUTE_CONFIGURE,
  VE_AUDIO_CONTROL_TYPE_SELECT_PRE_PROCESSOR_DATA_END_POINT,
  VE_AUDIO_CONTROL_TYPE_SELECT_POST_PROCESSOR_DATA_END_POINT,
}t_ve_audio_control_type_e;
```

The enumeration elements of **t_ve_audio_control_type_e** are as follows:

Element	Definition
VE_AUDIO_CONTROL_TYPE_ RECORD_START	Start recording the audio stream to a file. Selecting this enumeration type within **VE_ControlAudio** allows access to the **t_ve_audio_control_record_start_s** structure contained within the **t_ve_audio_control_u** union as described in Appendix B.31 on Page 343.
VE_AUDIO_CONTROL_TYPE_ RECORD_STOP	Stop recording the audio stream to a file. Selecting this enumeration type within **VE_ControlAudio** means that the **t_ve_audio_control_u** union as described in Appendix B.31 on Page 343, should be NULL.
VE_AUDIO_CONTROL_TYPE_ PLAY_ONCE_START	Start playing into the audio stream from a file. Selecting this enumeration type within **VE_ControlAudio** allows access to the **t_ve_audio_control_play_start_s** structure contained within the **t_ve_audio_control_u** union as described in Appendix B.31 on Page 343.
VE_AUDIO_CONTROL_TYPE_ PLAY_LOOP_START	Start playing into the audio stream from a file, looping the file until VE_AUDIO_CONTROL_TYPE_PLAY_STOP is used. Selecting this enumeration type within **VE_ControlAudio** allows access to the **t_ve_audio_control_play_start_s** structure contained within the **t_ve_audio_control_u** union as described in Appendix B.31 on Page 343.
VE_AUDIO_CONTROL_TYPE_ PLAY_STOP	Stop playing into the audio stream from a file. Selecting this enumeration type within **VE_ControlAudio** means that the **t_ve_audio_control_u** union as described in Appendix B.31 on Page 343, should be NULL.
VE_AUDIO_CONTROL_TYPE_ AUDIO_LIP_SYNC_ADJUSTMENT	Adjust the lip synchronization on this audio channel. Selecting this enumeration type within **VE_ControlAudio** allows access to the **t_ve_audio_control_lip_sync_adjustment_s** structure contained within the **t_ve_audio_control_u** union as described in Appendix B.31 on Page 343.

Continued >>

Element	Definition
VE_AUDIO_CONTROL_TYPE_ MUTE_CONFIGURE	Configure the audio mute for this channel. Selecting this enumeration type within **VE_ControlAudio** allows access to the **t_ve_audio_control_data_audio_mute_s** structure contained within the **t_ve_audio_control_u** union as described in Appendix B.31 on Page 343.
VE_AUDIO_CONTROL_TYPE_SELECT_ PRE_PROCESSOR_DATA_END_POINT	Change the pre-processor data-end point instance for this audio channel. Selecting this enumeration type within **VE_ControlAudio** allows access to the **t_ve_audio_control_data_end_point_selector_s** structure contained within the **t_ve_audio_control_u** union as described in Appendix B.31 on Page 343.
VE_AUDIO_CONTROL_TYPE_SELECT_ POST_PROCESSOR_DATA_END_POINT	Change the post-processor data-end point instance for this audio channel. Selecting this enumeration type within **VE_ControlAudio** allows access to the **t_ve_audio_control_data_end_point_selector_s** structure contained within the **t_ve_audio_control_u** union as described in Appendix B.31 on Page 343.

B.144 t_ve_audio_control_direction_e

The **t_ve_audio_control_direction_e** enumeration is used by the **t_ve_audio_control_play_start_s** structure, as described in Appendix B.31.1 on page 344. This control structure determines the direction of the audio stream that is to be played into or recorded from. This enumeration is defined in **ve_control_audio.h** and is described below:

```
typedef enum{
  VE_CONTROL_DIRECTION_AUDIO = 0,
  VE_CONTROL_DIRECTION_PACKET,
  VE_CONTROL_DIRECTION_PACKET_AND_AUDIO
}t_ve_audio_control_direction_e;
```

The enumeration elements are as follows:

Element	Definition
VE_CONTROL_DIRECTION_AUDIO	Control direction is for audio side only.
VE_CONTROL_DIRECTION_PACKET	Control direction is for packet side only.
VE_CONTROL_DIRECTION_PACKET_AND_AUDIO	Control direction is for both the audio and packet sides.

B.145 t_ve_audio_control_record_data_format_e

The **t_ve_audio_control_record_data_format_e** enumeration is used by the **t_ve_audio_control_record_start_s** structure as described in Appendix B.31.2 on Page 344. The **t_ve_audio_control_record_data_format_e** enumeration determines the format of the audio file to be recorded from the audio stream. This enumeration is defined in **ve_control_audio.h** and is described below:

```
typedef enum{
  VE_CONTROL_RECORD_RAW_16BIT_BYTE_REVERSED_STEREO
}t_ve_audio_control_record_data_format_e;
```

The enumeration element is as follows:

Element	Definition
VE_CONTROL_RECORD_RAW_16BIT_BYTE_REVERSED_STEREO	The record audio data format is to be recorded in 16-bit, byte-reversed stereo raw PCM.

B.146 t_ve_audio_control_play_data_format_e

The **t_ve_audio_control_play_data_format_e** enumeration is used by the **t_ve_audio_control_play_start_s** structure, as described in Appendix B.31.1 on page 344. The **t_ve_audio_control_play_data_format_e** enumeration determines the format of the audio file to be played into the audio stream. This enumeration is defined in **ve_control_audio.h** and is described below:

```
typedef enum{
  VE_CONTROL_PLAY_RAW_16BIT_BYTE_REVERSED_MONO,
  VE_CONTROL_PLAY_RAW_16BIT_BYTE_REVERSED_STEREO_LEFT,
  VE_CONTROL_PLAY_RAW_16BIT_BYTE_REVERSED_STEREO_RIGHT
}t_ve_audio_control_play_data_format_e;
```

The enumeration elements are as follows:

Element	Definition
VE_CONTROL_PLAY_RAW_16BIT_BYTE_REVERSED_MONO	The audio data format is to be played in 16-bit, byte-reversed mono raw PCM.
VE_CONTROL_PLAY_RAW_16BIT_BYTE_REVERSED_STEREO_LEFT	The audio data format for the left side (local) of a stereo channel is to be played in 16-bit, byte-reversed mono raw PCM.
VE_CONTROL_PLAY_RAW_16BIT_BYTE_REVERSED_STEREO_RIGHT	The audio data format for the right side (remote) of a stereo channel is to be played in 16-bit, byte-reversed mono raw PCM.

B.147 t_ve_audio_control_data_audio_mute_state_e

The **t_ve_audio_control_data_audio_mute_state_e** enumeration is used by the **t_ve_audio_control_data_audio_mute_s** structure, as described in Appendix B.31.5 on page 346. The **t_ve_audio_control_data_audio_mute_state_e** enumeration determines the mute state of the audio stream. This enumeration is defined in **ve_control_audio.h** and is described below:

```
typedef enum{
  VE_AUDIO_CONTROL_DATA_AUDIO_MUTE_STATE_ILLEGAL,
  VE_AUDIO_CONTROL_DATA_AUDIO_ACTIVATE_MUTE_STATE,
  VE_AUDIO_CONTROL_DATA_AUDIO_DEACTIVATE_MUTE_STATE,
  VE_AUDIO_CONTROL_DATA_AUDIO_MUTE_STATE_COUNT
}t_ve_audio_control_data_audio_mute_state_e;
```

The enumeration elements are as follows:

Element	Definition
VE_AUDIO_CONTROL_DATA_AUDIO_MUTE_STATE_ILLEGAL	This is an illegal mute state and is used internally by VeriCall Edge, it should not be used by the user application.
VE_AUDIO_CONTROL_DATA_AUDIO_ACTIVATE_MUTE_STATE	Enable mute of the audio stream.
VE_AUDIO_CONTROL_DATA_AUDIO_DEACTIVATE_MUTE_STATE	Disable mute of the audio stream.
VE_AUDIO_CONTROL_DATA_AUDIO_MUTE_STATE_COUNT	The number of states available in this enumeration. This element is used internally by VeriCall Edge and should not be used by the user application.

B.148 t_ve_audio_control_data_audio_mute_direction_e

The **t_ve_audio_control_data_audio_mute_direction_e** enumeration is used by the **t_ve_audio_control_data_audio_mute_s** structure, as described in Appendix B.31.5 on page 346. The **t_ve_audio_control_data_audio_mute_state_e** enumeration determines the direction of the mute operation on the audio stream. This enumeration is defined in **ve_control_audio.h** and is described below:

```
typedef enum{
  VE_AUDIO_CONTROL_DATA_AUDIO_MUTE_DIRECTION_ILLEGAL,
  VE_AUDIO_CONTROL_DATA_AUDIO_MUTE_DIRECTION_TX,
  VE_AUDIO_CONTROL_DATA_AUDIO_MUTE_DIRECTION_RX,
  VE_AUDIO_CONTROL_DATA_AUDIO_MUTE_DIRECTION_COUNT
}t_ve_audio_control_data_audio_mute_direction_e;
```

The enumeration elements are as follows:

Element	Definition
VE_AUDIO_CONTROL_DATA_AUDIO_MUTE_DIRECTION_ILLEGAL	This is an illegal state and is used internally by VeriCall Edge, it should not be used by the user application.
VE_AUDIO_CONTROL_DATA_AUDIO_MUTE_DIRECTION_TX	Audio mute operations will operate on the VeriCall Edge transmitted audio stream.
VE_AUDIO_CONTROL_DATA_AUDIO_MUTE_DIRECTION_RX	Audio mute operations will operate on the VeriCall Edge received audio stream.
VE_AUDIO_CONTROL_DATA_AUDIO_MUTE_DIRECTION_COUNT	The number of states available in this enumeration. This element is used internally by VeriCall Edge and should not be used by the user application.

B.149 t_ve_video_control_type_e

The **t_ve_video_control_type_e** enumeration is used by the **VE_ControlVideo** function to manipulate and control the local and received video image. This enumeration is defined in **ve_control_video.h** and is described below:

```
typedef enum{
 VE_VIDEO_CONTROL_TYPE_INVALID_SELECTION = 0,
 VE_VIDEO_CONTROL_TYPE_FILTER_SELECTION,
 VE_VIDEO_CONTROL_TYPE_SELECT_LOCAL_IMAGE_DISPLAY_STATE,
 VE_VIDEO_CONTROL_TYPE_SCALE_LOCAL_IMAGE,
 VE_VIDEO_CONTROL_TYPE_SCALE_RX_IMAGE,
 VE_VIDEO_CONTROL_TYPE_ROTATE_TX_IMAGE,
 VE_VIDEO_CONTROL_TYPE_ROTATE_LOCAL_IMAGE,
 VE_VIDEO_CONTROL_TYPE_ROTATE_RX_IMAGE,
 VE_VIDEO_CONTROL_TYPE_MIRROR_TX_IMAGE,
 VE_VIDEO_CONTROL_TYPE_MIRROR_LOCAL_IMAGE,
 VE_VIDEO_CONTROL_TYPE_MIRROR_RX_IMAGE,
 VE_VIDEO_CONTROL_TYPE_VIDEO_MUTE,
 VE_VIDEO_CONTROL_TYPE_MOVE_AND_RESIZE_LOCAL_IMAGE,
 VE_VIDEO_CONTROL_TYPE_MOVE_AND_RESIZE_RX_IMAGE,
 VE_VIDEO_CONTROL_TYPE_ENCODE_LOCAL_IMAGE,
 VE_VIDEO_CONTROL_TYPE_ENCODE_RX_IMAGE,
 VE_VIDEO_CONTROL_TYPE_LIP_SYNC_ADJUSTMENT,
 VE_VIDEO_CONTROL_TYPE_LOCAL_IMAGE_DISPLAY_DATA_END_POINT_SELECTOR,
 VE_VIDEO_CONTROL_TYPE_LOCAL_IMAGE_SOURCE_DATA_END_POINT_SELECTOR,
 VE_VIDEO_CONTROL_TYPE_RX_IMAGE_DISPLAY_DATA_END_POINT_SELECTOR,
 VE_VIDEO_CONTROL_TYPE_FREEZE_RX_IMAGE,
 VE_VIDEO_CONTROL_TYPE_RECONFIGURE_PRE_PROC_VIDEO_PARAMETERS,
 VE_VIDEO_CONTROL_TYPE_RECONFIGURE_POST_PROC_VIDEO_PARAMETERS,
 VE_VIDEO_CONTROL_TYPE_VIDEO_NUM_SELECTIONS
}t_ve_video_control_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_VIDEO_CONTROL_TYPE_ INVALID_SELECTION	This is an invalid control type selection.
VE_VIDEO_CONTROL_TYPE_ FILTER_SELECTION	Apply the filter control selection.
VE_VIDEO_CONTROL_TYPE_ SELECT_LOCAL_IMAGE_DISPLAY_STATE	Enable or disable the local video image display.
VE_VIDEO_CONTROL_TYPE_ SCALE_LOCAL_IMAGE	Scale the local video image.
VE_VIDEO_CONTROL_TYPE_ SCALE_RX_IMAGE	Scale the received video image.
VE_VIDEO_CONTROL_TYPE_ ROTATE_TX_IMAGE	Rotate the image to be transmitted.
VE_VIDEO_CONTROL_TYPE_ ROTATE_LOCAL_IMAGE	Rotate the local image.
VE_VIDEO_CONTROL_TYPE_ ROTATE_RX_IMAGE	Rotate the received video image.

Continued >>

Element	Definition
VE_VIDEO_CONTROL_TYPE_ MIRROR_TX_IMAGE	Mirror the video image to be transmitted.
VE_VIDEO_CONTROL_TYPE_ MIRROR_LOCAL_IMAGE	Mirror the local video image.
VE_VIDEO_CONTROL_TYPE_ MIRROR_RX_IMAGE	Mirror the received video image.
VE_VIDEO_CONTROL_TYPE_ VIDEO_MUTE	Mute the video image.
VE_VIDEO_CONTROL_TYPE_ MOVE_AND_RESIZE_LOCAL_IMAGE	Move and resize the local video image.
VE_VIDEO_CONTROL_TYPE_ MOVE_AND_RESIZE_RX_IMAGE	Move and resize the received video image.
VE_VIDEO_CONTROL_TYPE_ ENCODE_LOCAL_IMAGE	Encode the video image.
VE_VIDEO_CONTROL_TYPE_ ENCODE_RX_IMAGE	Encode the received image.
VE_VIDEO_CONTROL_TYPE_ LIP_SYNC_ADJUSTMENT	Apply lip-synchronization to the associated audio and video channel pair.
VE_VIDEO_CONTROL_TYPE_LOCAL_IMAGE_ DISPLAY_DATA_END_POINT_SELECTOR	Select the terminating data-end point for the local video image.
VE_VIDEO_CONTROL_TYPE_LOCAL_IMAGE_ SOURCE_DATA_END_POINT_SELECTOR	Select the source data-end point for the local video image.
VE_VIDEO_CONTROL_TYPE_RX_IMAGE_ DISPLAY_DATA_END_POINT_SELECTOR	Select the terminating data-end point for the received video image.
VE_VIDEO_CONTROL_TYPE_ FREEZE_RX_IMAGE	Locally freeze the received image.
VE_VIDEO_CONTROL_TYPE_RECONFIGURE_ PRE_PROC_VIDEO_PARAMETERS	Alter the video pre-processor parameters.
VE_VIDEO_CONTROL_TYPE_RECONFIGURE_ POST_PROC_VIDEO_PARAMETERS,	Alter the video post-processor parameters.
VE_VIDEO_CONTROL_TYPE_ VIDEO_NUM_SELECTIONS	The number of control type elements in this enumeration.

B.150 t_ve_video_control_filter_selector_e

The **t_ve_video_control_filter_selector_e** enumeration is used by **t_ve_video_control_u** as defined in Appendix B.32 on Page 347. The enumerated types in **t_ve_video_control_filter_selector_e** allow the selection of filtering to be performed on the video image. This enumeration is defined in **ve_control_video.h** and is described below:

```
typedef enum{
  VE_VIDEO_CONTROL_FILTER_SELECTION_ALL_DISABLE = 0,
  VE_VIDEO_CONTROL_FILTER_SELECTION_DERING_ENABLE,
  VE_VIDEO_CONTROL_FILTER_SELECTION_DEBLOCK_ENABLE,
  VE_VIDEO_CONTROL_FILTER_SELECTION_ALL_ENABLE,
}t_ve_video_control_filter_selector_e;
```

The enumeration elements of **t_ve_video_control_filter_selector_e** are as follows:

Element	Definition
VE_VIDEO_CONTROL_FILTER_SELECTION_ALL_DISABLE	Disable filtering of the video image.
VE_VIDEO_CONTROL_FILTER_SELECTION_DERING_ENABLE	Enable deringing of the video image.
VE_VIDEO_CONTROL_FILTER_SELECTION_DEBLOCK_ENABLE	Enable deblocking of the video image.
VE_VIDEO_CONTROL_FILTER_SELECTION_ALL_ENABLE	Apply all filtering to the video image (deringing and deblocking).

B.151 t_ve_video_control_select_display_state_e

The **t_ve_video_control_select_display_state_e** enumeration is used by **t_ve_video_control_u** as defined in Appendix B.32 on Page 347. The enumerated types in **t_ve_video_control_select_display_state_e** allow the ennoblement or disablement of the local video image. This enumeration is defined in **ve_control_video.h** and is described below:

```
typedef enum{
  VE_VIDEO_CONTROL_DISPLAY_STATE_DISABLE = 0,
  VE_VIDEO_CONTROL_DISPLAY_STATE ENABLE
}t_ve_video_control_select_display_state_e;
```

The enumeration elements are as follows:

Element	Definition
VE_VIDEO_CONTROL_DISPLAY_STATE_DISABLE	Disable the local video display.
VE_VIDEO_CONTROL_DISPLAY_STATE_ENABLE	Enable the local video display.

B.152 t_ve_video_control_rotate_e

The **t_ve_video_control_rotate_e** enumeration is used by **t_ve_video_control_u**, as described in Appendix B.32 on Page 347, as well as the **t_ve_video_control_reset_pre_proc_params_s** structure, which is described in Appendix B.32.7 on Page 352. The **t_ve_video_control_rotate_e** enumeration determines the clockwise rotation to be performed on the video image in degrees. The enumeration is defined in **ve_control_video.h** and is described below:

```
typedef enum{
  VE_VIDEO_CONTROL_IMAGE_ROTATE_0_DEGREES = 0;
  VE_VIDEO_CONTROL_IMAGE_ROTATE_90_DEGREES,
  VE_VIDEO_CONTROL_IMAGE_ROTATE_180_DEGREES,
  VE_VIDEO_CONTROL_IMAGE_ROTATE_270_DEGREES,
}t_ve_video_control_rotate_e;
```

The enumeration elements are as follows:

Element	Definition
VE_VIDEO_CONTROL_IMAGE_ROTATE_0_DEGREES	No rotation is required of the current image.
VE_VIDEO_CONTROL_IMAGE_ROTATE_90_DEGREES	Rotate the image clockwise by 90°.
VE_VIDEO_CONTROL_IMAGE_ROTATE_180_DEGREES	Rotate the image clockwise by 180°.
VE_VIDEO_CONTROL_IMAGE_ROTATE_270_DEGREES	Rotate the image clockwise by 270°.

B.153 t_ve_video_control_mirror_e

The **t_ve_video_control_mirror_e** enumeration is used by **t_ve_video_control_u**, as described in Appendix B.32 on Page 347, as well as the **t_ve_video_control_reset_pre_proc_params_s** structure, which is described in Appendix B.32.7 on Page 352. The **t_ve_video_control_mirror_e** enumeration determines the plane the image to be mirrored on. The enumeration is defined in **ve_control_video.h** and is described below:

```
typedef enum{
  VE_VIDEO_CONTROL_IMAGE_MIRROR_DISABLE = 0,
  VE_VIDEO_CONTROL_IMAGE_MIRROR_ENABLE_H_PLANE,
  VE_VIDEO_CONTROL_IMAGE_MIRROR_ENABLE_V_PLANE,
  VE_VIDEO_CONTROL_IMAGE_MIRROR_ENABLE_H_AND_V_PLANES;
}t_ve_video_control_mirror_e;
```

Continued >>

The enumeration elements of **t_ve_video_control_mirror_e** are as follows:

Element	Definition
VE_VIDEO_CONTROL_IMAGE_MIRROR_DISABLE	The image is not mirrored.
VE_VIDEO_CONTROL_IMAGE_MIRROR_ENABLE_H_PLANE	The image is mirrored on the horizontal plane only.
VE_VIDEO_CONTROL_IMAGE_MIRROR_ENABLE_V_PLANE	The image is mirrored on the vertical plane only.
VE_VIDEO_CONTROL_IMAGE_MIRROR_ENABLE_H_AND_V_PLANES	The image is mirrored on the horizontal and vertical plane.

B.154 t_ve_video_control_mute_mode_e

The **t_ve_video_control_mute_mode_e** enumeration is used by the **t_ve_video_control_mute_s** structure as defined in Appendix B.32.2 on Page 349. The **t_ve_video_control_mute_mode_e** enumeration determines the mute mode of operation to be performed on the video stream. The enumeration is defined in **ve_control_video.h** and is described below:

```
typedef enum{
  VE_VIDEO_CONTROL_MUTE_FILE_IMAGE = 0,
  VE_VIDEO_CONTROL_MUTE_LAST_FRAME,
  VE_VIDEO_CONTROL_MUTE_OFF
}t_ve_video_control_mute_mode_e;
```

The enumeration elements are as follows:

Element	Definition
VE_VIDEO_CONTROL_MUTE_FILE_IMAGE	Mute the video stream and send a file image to the stream.
VE_VIDEO_CONTROL_MUTE_LAST_FRAME	Mute the video stream and transmit the last video frame.
VE_VIDEO_CONTROL_MUTE_OFF	Disable a previously set mute mode.

To change any of the mute states, `VE_VIDEO_CONTROL_MUTE_OFF` must be called first. For example, to change from `VE_VIDEO_CONTROL_MUTE_FILE_IMAGE` to `VE_VIDEO_CONTROL_MUTE_LAST_FRAME` requires a `VE_VIDEO_CONTROL_MUTE_OFF` call first.

B.155 t_ve_video_control_image_freeze_state_e

The **t_ve_video_control_image_freeze_state_e** enumeration is used by **t_ve_video_control_u** as defined in Appendix B.32 on Page 347. The **t_ve_video_control_image_freeze_state_e** enumeration determines whether the received video image should be frozen or not. The enumeration is defined in **ve_control_video.h** and is described below:

```
typedef enum{
  VE_VIDEO_CONTROL_IMAGE_FREEZE_STATE_DISABLE = 0;
  VE_VIDEO_CONTROL_IMAGE_FREEZE_STATE_ENABLE
}t_ve_video_control_image_freeze_state_e;
```

The enumeration elements are as follows:

Element	Definition
VE_VIDEO_CONTROL_IMAGE_FREEZE_STATE_DISABLE	The received image should not be frozen.
VE_VIDEO_CONTROL_IMAGE_FREEZE_STATE_ENABLE	The received image should be frozen.

B.156 t_ve_video_control_encode_format_e

The **t_ve_video_control_encode_format_e** enumeration is used by the **t_ve_video_control_image_encode_s** structure as described in Appendix B.32.4 on Page 350. The **t_ve_video_control_encode_format_e** enumeration determines the video encode format for capturing a single frame to a file. The enumeration is defined in **ve_control_video.h** and is described below:

```
typedef enum{
  VE_VIDEO_CONTROL_ENCODE_FORMAT_JPEG
}t_ve_video_control_encode_format_e;
```

The enumeration element is as follows:

Element	Definition
VE_VIDEO_CONTROL_ENCODE_FORMAT_JPEG	The video encode format is JPEG.

B.157 t_ve_video_config_error_reason_e

The **t_ve_video_config_error_reason_e** enumeration is used by the **t_ve_user_msg_video_encode_path_config_err_s** structure, as described in Appendix B.33.59 on Page 409. The **t_ve_video_config_error_reason_e** enumeration determines the reason why the video configuration was rejected by VeriCall Edge. The enumeration is defined in **ve_events.h** and is described below:

```
typedef enum{
  VE_VIDEO_CONFIG_ERROR_REASON_INCOMPATIBLE_SIZE,
  VE_VIDEO_CONFIG_ERROR_REASON_ALL_PARAMETERS_ZERO
}t_ve_video_config_error_reason_e;
```

The enumeration elements are as follows:

Element	Definition
VE_VIDEO_CONFIG_ERROR_REASON_INCOMPATIBLE_SIZE	The configuration structure size is not compatible with that expected by VeriCall Edge.
VE_VIDEO_CONFIG_ERROR_REASON_ALL_PARAMETERS_ZERO	All the configuration entries were zero.

B.158 t_ve_alarm_type_e

The **t_ve_alarm_type_e** enumeration is part of the **t_ve_alarm_s** structure defined in Appendix B.33.58 on Page 408. The **t_ve_alarm_type_e** enumeration determines the VeriCall Edge module that derived the alarm. The **t_ve_alarm_type_e** enumeration is defined in **ve_alarms.h** and is described below:

```
typedef enum{
  VE_ALARM_TYPE_SYSTEM,
  VE_ALARM_TYPE_SIP,
  VE_ALARM_TYPE_H323
}t_ve_alarm_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_ALARM_TYPE_SYSTEM	The alarm was a system alarm. A system alarm can be generated from the VeriCall Edge scheduler, data manager or algorithm.
VE_ALARM_TYPE_SIP	The alarm was generated from the SIP stack.
VE_ALARM_TYPE_H323	The alarm was generated from the H.323 stack.

B.159 t_ve_alarm_system_subtype_e

The **t_ve_alarm_system_subtype_e** enumeration is part of the **t_ve_alarm_s** structure defined in Appendix B.33.58 on Page 408. The **t_ve_alarm_system_subtype_e** enumeration determines the subtype of alarm. The **t_ve_alarm_system_subtype_e** enumeration is defined in **ve_alarms.h** and is described below:

```
typedef enum{
  VE_ALARM_SYSTEM_SUBTYPE_NONE = 0,
  VE_ALARM_SYSTEM_SUBTYPE_AJBM_DISCARD_DUE_TO_BUFFER_FULL_SET,
  VE_ALARM_SYSTEM_SUBTYPE_AJBM_DISCARD_DUE_TO_BUFFER_FULL_CLEAR,
  VE_ALARM_SYSTEM_SUBTYPE_AJBM_EXTENDED_BREAK_IN_STREAM_SET,
  VE_ALARM_SYSTEM_SUBTYPE_AJBM_EXTENDED_BREAK_IN_STREAM_CLEAR,
  VE_ALARM_SYSTEM_SUBTYPE_AJBM_UNDERRUN_SET,
  VE_ALARM_SYSTEM_SUBTYPE_AJBM_UNDERRUN_CLEAR,
  VE_NUM_ALARM_SYSTEM_SUBTYPES
}t_ve_alarm_system_subtype_e;
```

Continued >>

The enumeration elements of **t_ve_alarm_system_subtype_e** are as follows:

Element	Definition
VE_ALARM_SYSTEM_SUBTYPE_NONE	There is no alarm subtype associated with the generated alarm.
VE_ALARM_SYSTEM_SUBTYPE_AJBM_ DISCARD_DUE_TO_BUFFER_FULL_SET	The AJBM raised an alarm due to its internal buffer being full.
VE_ALARM_SYSTEM_SUBTYPE_AJBM_ DISCARD_DUE_TO_BUFFER_FULL_CLEAR	The AJBM is no longer discarding packets due to its internal buffer being full.
VE_ALARM_SYSTEM_SUBTYPE_AJBM_ EXTENDED_BREAK_IN_STREAM_SET	The AJBM has reported an extended break in its received packet stream.
VE_ALARM_SYSTEM_SUBTYPE_AJBM_ EXTENDED_BREAK_IN_STREAM_CLEAR	The AJBM is no longer detecting an extended break in its received packet stream.
VE_ALARM_SYSTEM_SUBTYPE_AJBM_ UNDERRUN_SET	The AJBM has reported an under run alarm.
VE_ALARM_SYSTEM_SUBTYPE_AJBM_ UNDERRUN_CLEAR,	The AJBM is no longer detecting an under run alarm situation.
VE_NUM_ALARM_SYSTEM_SUBTYPES	This enumerated entry should equal the sum of available system alarm subtypes (both set and clear entries).

B.160 t_ve_alarm_sip_subtype_e

The **t_ve_alarm_sip_subtype_e** enumeration is used by the **t_ve_user_msg_alarm_s** structure as described in Appendix B.33.58 on Page 408. The enumerations of **t_ve_alarm_sip_subtype_e** determine the subtype of the SIP alarm. This enumeration is defined in **ve_sip_alarms.h** and is described below:

```
typedef enum{
  VE_ALARM_SIP_SUBTYPE_NONE=0,
  VE_ALARM_SIP_SUBTYPE_INITIALIZE_FAILED,
  VE_ALARM_SIP_SUBTYPE_INITIALIZE_OPEN_LISTEN_PORT_FAILED,
  VE_ALARM_SIP_SUBTYPE_PROCESS_FAILED,
  VE_ALARM_SIP_SUBTYPE_INTERNAL_ERROR,
  VE_NUM_ALARM_SIP_SUBTYPES
}t_ve_alarm_sip_subtype_e;
```

Continued >>

The enumeration elements of **t_ve_alarm_sip_subtype_e** are as follows:

Element	Definition
VE_ALARM_SIP_SUBTYPE_NONE	There was no specific alarm subtype identified.
VE_ALARM_SIP_SUBTYPE_ INITIALIZE_FAILED	The SIP stack failed to initialize.
VE_ALARM_SIP_SUBTYPE_ INITIALIZE_OPEN_LISTEN_PORT_FAILED	The SIP stack failed to open the specified SIP port number.
VE_ALARM_SIP_SUBTYPE_ PROCESS_FAILED	A SIP stack process failed for an unknown reason.
VE_ALARM_SIP_SUBTYPE_ INTERNAL_ERROR	There was a SIP stack internal error.
VE_NUM_ALARM_SIP_SUBTYPES	The number of alarm types defined in this enumerated type.

B.161 t_ve_alarm_h323_subtype_e

The **t_ve_alarm_h323_subtype_e** enumeration is used by the **t_ve_user_msg_alarm_s** structure as described in Appendix B.33.58 on Page 408. The enumerations of **t_ve_alarm_h323_subtype_e** determine the subtype of the H.323 alarm. This enumeration is defined in **ve_h323_alarms.h** and is described below:

```
typedef enum{
  VE_ALARM_H323_SUBTYPE_NONE = 0,
  VE_ALARM_H323_SUBTYPE_INTERNAL_ERROR,
  VE_NUM_ALARM_H323_SUBTYPES
}t_ve_alarm_h323_subtype_e;
```

The enumeration element is as follows:

Element	Definition
VE_ALARM_H323_SUBTYPE_NONE	There was no specific alarm subtype identified.
VE_ALARM_H323_SUBTYPE_INTERNAL_ERROR	There was an unspecified internal H.323 error.
VE_NUM_ALARM_H323_SUBTYPES	The number of H.323 alarm subtype elements in this enumeration.

B.162 t_ve_user_msg_type_e

The **t_ve_user_msg_type_e** enumeration is part of the **t_ve_user_msg_s** structure as described in Appendix B.33 on Page 356. The **t_ve_user_msg_type_e** enumeration type contains the type of message received. The **t_ve_user_msg_type_e** enumeration is defined in **ve_events.h** and is described below:

```
typedef enum{
  VE_USER_MSG_NAT_MSG_ID_START,
  VE_USER_MSG_NAT_DISCOVER_RESPONSE = VE_USER_MSG_NAT_MSG_ID_START,
  VE_USER_MSG_NAT_USER_ADDRESSES_RESPONSE,
  VE_USER_MSG_NAT_REQUEST_FAILED,
  VE_USER_MSG_NAT_KEEP_ALIVE_EVENT,

  VE_USER_MSG_SIP_MSG_ID_START,
  VE_USER_MSG_SIP_SESSION_STARTED = VE_USER_MSG_SIP_MSG_ID_START,
  VE_USER_MSG_SIP_SESSION_UPDATED,
  VE_USER_MSG_SIP_SESSION_STOPPED,
  VE_USER_MSG_SIP_INVITE_RX,
  VE_USER_MSG_SIP_REINVITE_RX,
  VE_USER_MSG_SIP_INVITE_RESPONSE_RX,
  VE_USER_MSG_SIP_REGISTER_RESPONSE_RX,
  VE_USER_MSG_SIP_REFER_RX,
  VE_USER_MSG_SIP_REFER_RESPONSE_RX,
  VE_USER_MSG_SIP_REFER_NOTIFY_RX,
  VE_USER_MSG_SIP_UPDATE_RX,
  VE_USER_MSG_SIP_UPDATE_RESPONSE_RX,
  VE_USER_MSG_SIP_REINVITE_RESPONSE_RX,
  VE_USER_MSG_SIP_MESSAGE_RX,
  VE_USER_MSG_SIP_MESSAGE_RESPONSE_RX,
  VE_USER_MSG_SIP_OPTIONS_RX,
  VE_USER_MSG_SIP_OPTIONS_RESPONSE_RX,
  VE_USER_MSG_SIP_INFO_RX,
  VE_USER_MSG_SIP_INFO_RESPONSE_RX,
  VE_USER_MSG_SIP_SERVICE_RX,
  VE_USER_MSG_SIP_SERVICE_RESPONSE_RX,
  VE_USER_MSG_SIP_PRESENCE_SUBSCRIBE_RX,
  VE_USER_MSG_SIP_PRESENCE_NOTIFY_RX,
  VE_USER_MSG_SIP_PRESENCE_SUBSCRIBE_RESPONSE_RX,
  VE_USER_MSG_SIP_MW_NOTIFY_RX,
  VE_USER_MSG_SIP_MW_SUBSCRIBE_RESPONSE_RX,
  VE_USER_MSG_SIP_PRACK_RX,
  VE_USER_MSG_SIP_DIALOG_SUBSCRIBE_RX,
  VE_USER_MSG_SIP_DIALOG_NOTIFY_RX,
  VE_USER_MSG_SIP_DIALOG_SUBSCRIBE_RESPONSE_RX,
  VE_USER_MSG_SIP_OPERATION_FAILED,

  VE_USER_MSG_H323_MSG_ID_START,
  VE_USER_MSG_H323_REGISTRATION_RESPONSE = VE_USER_MSG_H323_
                                           MSG_ID_START,
  VE_USER_MSG_H323_CALL_STARTED,
  VE_USER_MSG_H323_CALL_STOPPED,
```

Continued >>

```
    VE_USER_MSG_H323_CALL_UPDATED,
    VE_USER_MSG_H323_CALL_REQUEST,
    VE_USER_MSG_H323_VIDEO_FEATURE_REQUEST,
    VE_USER_MSG_H323_CALL_RESPONSE,
    VE_USER_MSG_H323_TRANSFER_RESPONSE,
    VE_USER_MSG_H323_CALL_FORWARDED,
    VE_USER_MSG_H323_FORWARD_RESPONSE,
    VE_USER_MSG_H323_USER_INPUT_RECEIVED,
    VE_USER_MSG_H323_CALL_HOLD_RESPONSE,
    VE_USER_MSG_H323_MWI_RESPONSE,
    VE_USER_MSG_H323_MESSAGE_WAITING,
    VE_USER_MSG_H323_REQUEST_FAILED,

    VE_USER_MSG_MEDIA_MSG_ID_START,
    VE_USER_MSG_DETECT_DTMF_TONE = VE_USER_MSG_MEDIA_MSG_ID_START,
    VE_USER_MSG_DETECT_CP_TONE,
    VE_USER_MSG_DETECT_2100_TONE,
    VE_USER_MSG_DETECT_1100_TONE,
    VE_USER_MSG_RECEIVED_RFC2833_DIGIT,
    VE_USER_MSG_PLAY_PROMPT_EVENT,
    VE_USER_MSG_RECORD_PROMPT_EVENT,
    VE_USER_MSG_ENCODE_IMAGE_EVENT,
    VE_USER_MSG_ENCODE_PATH_CONFIG_ERROR,
    VE_USER_MSG_STREAM_EVENT,
    VE_USER_MSG_UNDECODEABLE_STREAM_EVENT
    VE_USER_MSG_DECODE_MEDIA_CHANGE_EVENT,
    VE_USER_MSG_NET_MSG_ID_START,
    VE_USER_MSG_NET_LOOKUP_RESPONSE = VE_USER_MSG_NET_
                                      MSG_ID_START,
    VE_USER_MSG_NET_REQUEST_FAILED,
    VE_USER_MSG_SYS_MSG_ID_START,
    VE_USER_MSG_MODULE_STATE_TRANSITION_EVENT =  VE_USER_MSG_SYS_
                                                 MSG_ID_START,
    VE_USER_MSG_ALARM,
    VE_USER_MSG_NETWORK_CONNECTION_EVENT,
    VE_USER_MSG_SHUTDOWN_COMPLETE,

    VE_USER_MSG_MSG_ID_END
}t_ve_user_msg_type_e;
```

The enumeration elements of **t_ve_user_msg_type_e** are as follows:

Element	Definition
VE_USER_MSG_NAT_MSG_ID_START	Identifies start of NAT messages.
VE_USER_MSG_NAT_DISCOVER_RESPONSE	Message to indicate a response to a NAT discover request.
VE_USER_MSG_NAT_USER_ADDRESSES_ RESPONSE	Response indicating a message relating to NAT user addresses.
VE_USER_MSG_NAT_REQUEST_FAILED	Response message indicating a NAT request has failed.
VE_USER_MSG_NAT_KEEP_ALIVE_EVENT	Message indicating a response relating to NAT keep-alive events.
VE_USER_MSG_SIP_MSG_ID_START	Identifies start of SIP messages.
VE_USER_MSG_SIP_SESSION_STARTED	Message response to indicate the start of a VeriCall Edge SIP session.

Element	Definition
VE_USER_MSG_SIP_SESSION_UPDATED	Message response to a SIP session update request.
VE_USER_MSG_SIP_SESSION_STOPPED	Message to indicate the end of a VeriCall Edge SIP session.
VE_USER_MSG_SIP_INVITE_RX	Message to indicate a SIP INVITE has been received.
VE_USER_MSG_SIP_REINVITE_RX	Message to indicate a SIP RE-INVITE message has been received.
VE_USER_MSG_SIP_INVITE_RESPONSE_RX	Message to indicate the response to a SIP INVITE message has been received.
VE_USER_MSG_SIP_REGISTER_RESPONSE_RX	Message to indicate the response to a SIP registration message has been received.
VE_USER_MSG_SIP_REFER_RX	Message to indicate a SIP REFER has been received.
VE_USER_MSG_SIP_REFER_RESPONSE_RX	Message to indicate the response to a SIP REFER message has been received.
VE_USER_MSG_SIP_REFER_NOTIFY_RX	Message to indicate a SIP NOTIFY has been received.
VE_USER_MSG_SIP_UPDATE_RX	Message to indicate a SIP UPDATE has been received.
VE_USER_MSG_SIP_UPDATE_RESPONSE_RX	Message to indicate a SIP UPDATE response has been received.
VE_USER_MSG_SIP_REINVITE_RESPONSE_RX	Message to indicate a SIP REINVITE response message has been received.
VE_USER_MSG_SIP_MESSAGE_RX	Message to indicate a SIP MESSAGE has been received.
VE_USER_MSG_SIP_MESSAGE_RESPONSE_RX	Message to indicate a SIP MESSAGE response has been received.
VE_USER_MSG_SIP_OPTIONS_RX	Message to indicate a SIP OPTIONS has been received.
VE_USER_MSG_SIP_OPTIONS_RESPONSE_RX	Message to indicate a SIP OPTIONS response has been received.
VE_USER_MSG_SIP_INFO_RX	Message to indicate a SIP INFO has been received.
VE_USER_MSG_SIP_INFO_RESPONSE_RX	Message to indicate a SIP INFO response has been received.
VE_USER_MSG_SIP_SERVICE_RX	Message to indicate a SIP SERVICE has been received.
VE_USER_MSG_SIP_SERVICE_RESPONSE_RX	Message to indicate a SIP SERVICE response has been received.
VE_USER_MSG_SIP_PRESENCE_SUBSCRIBE_ RX	Message to indicate a SIP presence SUBCRIBE message has been received.
VE_USER_MSG_SIP_PRESENCE_NOTIFY_RX	Message to indicate a SIP presence NOTIFY has been received.
VE_USER_MSG_SIP_PRESENCE_SUBSCRIBE_ RESPONSE_RX	Message to indicate a SIP presence SUBCRIBE response has been received.
VE_USER_MSG_SIP_MW_NOTIFY_RX	Message to indicate a SIP Message Waiting NOTIFY message has been received.
VE_USER_MSG_SIP_MW_SUBSCRIBE_ RESPONSE_RX	Message to indicate a SIP Message Waiting NOTIFY response has been received.
VE_USER_MSG_SIP_PRACK_RX	Message to indicate a SIP PRACK message has been received.
VE_USER_MSG_SIP_DIALOG_SUBSCRIBE_RX	Message to indicate a SIP dialog SUBCRIBE has been received.
VE_USER_MSG_SIP_DIALOG_NOTIFY_RX	Message to indicate a SIP dialog NOTIFY has been received.
VE_USER_MSG_SIP_DIALOG_SUBSCRIBE_ RESPONSE_RX	Message to indicate a SIP dialog SUBCRIBE response has been received.
VE_USER_MSG_SIP_OPERATION_FAILED	Message to indicate a SIP operation has failed. This message will contain the failed SIP message and the relevant message data and failure reason.
VE_USER_MSG_H323_MSG_ID_START,	Identifies start of H323 messages.

Element	Definition
VE_USER_MSG_H323_REGISTRATION_ RESPONSE	Message to indicate a response from the H.323 server the user was registered with.
VE_USER_MSG_H323_CALL_STARTED	Message to indicate an H.323 call has started. The stream information negotiated by the H.323 stack is contained within the message and the event is generated for both incoming and outgoing calls.
VE_USER_MSG_H323_CALL_STOPPED	Message to indicate an H.323 call has been stopped.
VE_USER_MSG_H323_CALL_UPDATED	Message to indicate an H.323 call stream has been updated. The updated stream information is contained within the message.
VE_USER_MSG_H323_CALL_REQUEST	Message to indicate an incoming H.323 call request has been received.
VE_USER_MSG_H323_VIDEO_FEATURE_ REQUEST	Message to indicate a response to a H.323 video command.
VE_USER_MSG_H323_CALL_RESPONSE	Message to indicate a response has been received for an outbound H.323 call.
VE_USER_MSG_H323_TRANSFER_RESPONSE	Message to indicate a response to a call transfer call command.
VE_USER_MSG_H323_CALL_FORWARDED	Message to indicate a call has been forwarded.
VE_USER_MSG_H323_FORWARD_RESPONSE	Message to indicate a response to a call forward command.
VE_USER_MSG_H323_USER_INPUT_ RECEIVED	Message to indicate a character string via the H.323 plane has been received.
VE_USER_MSG_H323_CALL_HOLD_RESPONSE	Message to indicate a response to a call hold command.
VE_USER_MSG_H323_MWI_RESPONSE	Message to indicate a response to a MWI command.
VE_USER_MSG_H323_MESSAGE_WAITING	Message to indicate a H.323 call is waiting at the message center.
VE_USER_MSG_H323_REQUEST_FAILED	Message to indicate an API function call was legally made but was not completed by the H.323 stack. The message contains information for the cause of the failure.
VE_USER_MSG_MEDIA_MSG_ID_START	Identifies start of media messages.
VE_USER_MSG_DETECT_DTMF_TONE	Message to indicate the detection of a DTMF tone.
VE_USER_MSG_DETECT_CP_TONE	Message to indicate the detection of a Call Progress tone.
VE_USER_MSG_DETECT_2100_TONE	Message to indicate the detection of a 2100Hz tone.
VE_USER_MSG_DETECT_1100_TONE	Message to indicate the detection of a 1100Hz tone.
VE_USER_MSG_RECEIVED_RFC2833_DIGIT	Message to indicate a RFC2833 digit has been received.
VE_USER_MSG_PLAY_PROMPT_EVENT	Message to indicate a response to a play into audio stream request.
VE_USER_MSG_RECORD_PROMPT_EVENT	Message to indicate a response to a record from audio stream request.
VE_USER_MSG_ENCODE_IMAGE_EVENT	Message to indicate a response to a request to encode a video image to a defined format, e.g. JPEG.
VE_USER_MSG_ENCODE_PATH_CONFIG_ ERROR	Message to indicate an encoder configuration failure. The message contains information relating to this failure.
VE_USER_MSG_STREAM_EVENT	Message to indicate a stream event has occurred.
VE_USER_MSG_UNDECODEABLE_STREAM_ EVENT	Message to indicate an undecodable RTP stream has been received.
VE_USER_MSG_DECODE_MEDIA_CHANGE_ EVENT	Message to indicate a change event has been received for the decode media.

Element	Definition
VE_USER_MSG_NET_MSG_ID_START	Identifies start of NET messages.
VE_USER_MSG_NET_LOOKUP_RESPONSE	Message to indicate a response to a network lookup request.
VE_USER_MSG_NET_REQUEST_FAILED	Message to indicate a network lookup request has failed. The response contains information of the lookup request that failed and the failure reason.
VE_USER_MSG_SYS_MSG_ID_START	Identifies start of system messages.
VE_USER_MSG_MODULE_STATE_ TRANSITION_EVENT	Message to indicate a resource-usage module transition has been requested.
VE_USER_MSG_ALARM	Message to indicate a VeriCall Edge system alarm event.
VE_USER_MSG_NETWORK_ CONNECTION_EVENT	Message to indicate a network level connection event has occurred.
VE_USER_MSG_SHUTDOWN_COMPLETE	Message to indicate the request to shutdown the VeriCall Edge system was completed successfully.
VE_USER_MSG_MSG_ID_END	Identifies end of messages.

B.163 t_ve_user_msg_stream_event_reason_e

The **t_ve_user_msg_stream_event_reason_e** enumeration is used by the **t_ve_user_msg_stream_event_s** structure as described in Appendix B.33.60 on Page 411. The **t_ve_user_msg_stream_event_reason_e** enumeration type presents the reason for the stream event being raised. This enumeration is defined in **ve_events.h** and is described below:

```
typedef enum{
  VE_MSG_STREAM_EVENT_SUCCESS = 0,
  VE_MSG_STREAM_EVENT_ERROR_MODULE_IDLE_STATE,
  VE_MSG_STREAM_EVENT_ERROR_MODULE_ILLEGAL_CONFIG,
  VE_MSG_STREAM_EVENT_ERROR_MODULE_ILLEGAL_IP_ADDRESS,
  VE_MSG_STREAM_EVENT_ERROR_MODULE_ILLEGAL_CHANNEL,
  VE_MSG_STREAM_EVENT_ERROR_MODULE_PACKET,
  VE_MSG_STREAM_EVENT_FAILED_FAST_VIDEO_UPDATE,
  VE_MSG_STREAM_EVENT_INTERNAL_ERROR
}t_ve_user_msg_stream_event_reason_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MSG_STREAM_EVENT_SUCCESS	The start/update stream has been successful.
VE_MSG_STREAM_EVENT_ERROR_MODULE_IDLE_STATE	The start stream failed due to the module in the channel being in the idle state.
VE_MSG_STREAM_EVENT_ERROR_MODULE_ILLEGAL_CONFIG	The start/update stream message is illegal.
VE_MSG_STREAM_EVENT_ERROR_MODULE_ILLEGAL_IP_ADDRESS	The specified IP address is illegal.
VE_MSG_STREAM_EVENT_ERROR_MODULE_ILLEGAL_CHANNEL	The specified channel number is illegal.
VE_MSG_STREAM_EVENT_ERROR_MODULE_PACKET	The VeriCall Edge system failed to open or configure the packet interface.
VE_MSG_STREAM_EVENT_FAILED_FAST_VIDEO_UPDATE	The VeriCall Edge system failed to send a fast video update message. This entry is only valid for a video channel.
VE_MSG_STREAM_EVENT_INTERNAL_ERROR	A VeriCall Edge internal error was detected.

B.164 t_ve_user_msg_stream_event_type_e

The **t_ve_user_msg_stream_event_type_e** enumeration is used by the **t_ve_user_msg_stream_event_s** structure as described in Appendix B.33.60 on Page 411. The **t_ve_user_msg_stream_event_type_e** enumeration type presents the reason for the stream event being raised. This enumeration is defined in **ve_events.h** and is described below:

```
typedef enum{
  VE_MSG_STREAM_EVENT_TYPE_START_OR_UPDATE = 0,
  VE_MSG_STREAM_EVENT_TYPE_STOP
}t_ve_user_msg_stream_event_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_MSG_STREAM_EVENT_TYPE_START_OR_UPDATE	The event message is in response to a start stream or an update stream.
VE_MSG_STREAM_EVENT_TYPE_STOP	The event message is in response to a stop stream.

B.165 t_ve_user_msg_module_state_transition_reason_e

The **t_ve_user_msg_module_state_transition_reason_e** enumeration is used by the **t_ve_user_msg_module_state_transition_event_s** structure as described in Appendix B.33.61 on Page 412. The **t_ve_user_msg_module_state_transition_reason_e** enumeration type presents the reason for the module state transition failure. This enumeration is defined in **ve_events.h** and is described below:

```
typedef enum{
  VE_USER_MSG_STATE_TRANSITION_ERROR_VIDEO_ACTIVE,
  VE_USER_MSG_STATE_TRANSITION_ERROR_AUDIO_ACTIVE
}t_ve_user_msg_module_state_transition_reason_e;
```

The enumeration elements are as follows:

Element	Definition
VE_USER_MSG_STATE_TRANSITION_ERROR_VIDEO_ACTIVE	VeriCall Edge was unable to transition the video path to IDLE due to there currently being an active stream.
VE_USER_MSG_STATE_TRANSITION_ERROR_AUDIO_ACTIVE	VeriCall Edge was unable to transition the audio path to IDLE due to there currently being an active stream.

B.166 t_ve_sip_request_failed_reason_e

The **t_ve_sip_request_failed_reason_e** enumeration is part of the **t_ve_user_msg_sip_operation_failed_s** structure defined in Appendix B.33.26 on Page 383. The enumeration returns the reason of the failure. This enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_REQUEST_FAILED_INVALID_USER_HANDLE          = 0,
  VE_SIP_REQUEST_FAILED_INVALID_SIP_SESSION_HANDLE   = 1,
  VE_SIP_REQUEST_FAILED_INVALID_RESPONSE_CODE        = 2,
  VE_SIP_REQUEST_FAILED_NO_FREE_DIALOGS              = 3,
  VE_SIP_REQUEST_FAILED_NO_FREE_TRANSACTIONS         = 4,
  VE_SIP_REQUEST_FAILED_NO_FREE_TRANSACTION_USERS    = 5,
  VE_SIP_REQUEST_FAILED_NO_FREE_REGISTRATION_TIMERS  = 6,
  VE_SIP_REQUEST_FAILED_TRANSPORT_ERROR              = 7,
  VE_SIP_REQUEST_FAILED_FUNCTION_CALL_INVALID        = 8,
  VE_SIP_REQUEST_FAILED_INVALID_SIP_URI              = 9,
  VE_SIP_REQUEST_FAILED_CLIENT_TRANSACTION_TIMEOUT   = 10,
  VE_SIP_REQUEST_FAILED_NO_FREE_PRESENCE_USERS       = 11,
  VE_SIP_REQUEST_FAILED_NO_FREE_MW_USERS             = 12,
  VE_SIP_REQUEST_FAILED_PRACK_PENDING                = 13,
  VE_SIP_REQUEST_FAILED_PRACK_ERROR                  = 14,
  VE_SIP_REQUEST_FAILED_UNSUPPORTED_BY_DESTINATION   = 15,
  VE_SIP_REQUEST_FAILED_INTERNAL_ERROR               = 16
}t_ve_sip_request_failed_reason_e;
```

The enumeration elements of **t_ve_sip_request_failed_reason_e** are as follows:

Element	Definition
VE_SIP_REQUEST_FAILED_ INVALID_USER_HANDLE	The SIP stack has been passed an invalid user handle.
VE_SIP_REQUEST_FAILED_ INVALID_SIP_SESSION_HANDLE	The SIP stack has been passed an invalid SIP session handle.
VE_SIP_REQUEST_FAILED_ INVALID_RESPONSE_CODE	The SIP stack has been passed an invalid response code.
VE_SIP_REQUEST_FAILED_ NO_FREE_DIALOGS	The SIP stack has no free dialogs. Either more memory needs to be allocated at VeriCall Edge start-up or try again later when dialogs have become free.
VE_SIP_REQUEST_FAILED_ NO_FREE_TRANSACTIONS	The SIP stack has run out of free transactions. Either more memory needs to be allocated at VeriCall Edge start-up or try again later when transactions have become free.
VE_SIP_REQUEST_FAILED_ NO_FREE_TRANSACTION_USERS	The SIP stack has run out of free transactions. Either allocate more transactions to the SIP stack or try again later when transactions have been become free.
VE_SIP_REQUEST_FAILED_ NO_FREE_REGISTRATION_TIMERS	The SIP stack has run out of free registrations timers. Either more memory needs to be allocated at VeriCall Edge start-up or try again later when registration timers have become free.
VE_SIP_REQUEST_FAILED_ TRANSPORT_ERROR	The SIP stack encountered a transportation error because of the request.

Element	Definition
VE_SIP_REQUEST_FAILED_ FUNCTION_CALL_INVALID	The API function call made by the user application is invalid at this point in the SIP transaction. This is due to the caller trying to send a response, or the called trying to stop a SIP session before it is active
VE_SIP_REQUEST_FAILED_ INVALID_SIP_URI	The passed SIP URI was invalid.
VE_SIP_REQUEST_FAILED_ CLIENT_TRANSACTION_TIMEOUT	The client transaction timed out. This indicates the destination is present but did not response to the requests sent.
VE_SIP_REQUEST_FAILED_ NO_FREE_PRESENCE_USERS	The SIP stack has run out of free presence users. Either allocate more to the SIP stack or try again later when presence users have been become free.
VE_SIP_REQUEST_FAILED_ NO_FREE_MW_USERS	The SIP stack has run out of free message waiting subscription users. Check that the application is not attempting to subscribe multiple times for the same account mailbox rather than using a single subscription. If the user application requires to subscribe to many different account mailboxes, increase the SIP stack message waiting user allocation.
VE_SIP_REQUEST_FAILED_ PRACK_PENDING	An attempt has been made to send a response when PRACK has not been received for prior response.
VE_SIP_REQUEST_FAILED_ PRACK_ERROR	The PRACK mechanism has timed out or has detected an error.
VE_SIP_REQUEST_FAILED_ UNSUPPORTED_BY_DESTINATION	The requested operation is not supported by the destination and the SIP stack was not able to complete the request.
VE_SIP_REQUEST_FAILED_ INTERNAL_ERROR	An internal error has occurred in the SIP stack.

B.167 t_ve_user_sub_channel_select_e

The **t_ve_user_sub_channel_select_e** enumeration is used as an argument to the **VE_NATSetChannelAddressesData** and **VE_NATGetChannelAddressesData** API functions as well as the following structures:

- **t_ve_user_channel_selection_s** described in Appendix B.26.1.1 on page 301.
- **t_ve_media_description_s** described in Appendix B.27 on page 302.
- **t_ve_user_msg_detect_tone_dtmf_s** described in Appendix B.33.51 on page 404.
- **t_ve_user_msg_detect_tone_cp_s** described in Appendix B.33.52 on page 405.
- **t_ve_user_msg_detect_tone_2100_s** described in Appendix B.33.53 on page 405.
- **t_ve_user_msg_detect_tone_1100_s** described in Appendix B.33.54 on page 406.
- **t_ve_user_msg_play_prompt_event_s** described in Appendix B.33.56 on page 407.
- **t_ve_user_msg_record_prompt_event_s** described in Appendix B.33.57 on page 407.
- **t_ve_user_msg_stream_event_s** described in Appendix B.33.60 on page 411.
- **t_ve_user_msg_nat_user_addresses_response_s** described in Appendix B.33.2 on page 360.

The **t_ve_user_sub_channel_select_e** enumeration determines the VeriCall Edge sub-channel type to be used. This enumeration is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_USER_SUB_CHANNEL_SELECT_ILLEGAL,
  VE_USER_SUB_CHANNEL_SELECT_PRIMARY,
  VE_USER_SUB_CHANNEL_SELECT_SECONDARY,
  VE_USER_SUB_CHANNEL_SELECT_COUNT
}t_ve_user_sub_channel_select_e;
```

The enumeration elements are as follows:

Element	Definition
VE_USER_SUB_CHANNEL_SELECT_ILLEGAL	This element is used internally by VeriCall Edge and should not be used by the user application.
VE_USER_SUB_CHANNEL_SELECT_PRIMARY	The selected sub-channel has been designated as being a primary sub-channel.
VE_USER_SUB_CHANNEL_SELECT_SECONDARY	The selected sub-channel has been designated as being a secondary sub-channel.
VE_USER_SUB_CHANNEL_SELECT_COUNT	This element is used internally by VeriCall Edge and should not be used by the user application.

B.168 t_ve_user_sub_channel_index_e

The **t_ve_user_sub_channel_index_e** enumeration is reserved for future use and should not be used by the user application. This enumeration is currently defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_USER_SUB_CHANNEL_INDEX_PRIMARY,
  VE_USER_SUB_CHANNEL_INDEX_SECONDARY,
  VE_USER_SUB_CHANNEL_INDEX_COUNT
}t_ve_user_sub_channel_index_e;
```

The elements of this reserved enumeration are as follows:

Element	Definition
VE_USER_SUB_CHANNEL_INDEX_PRIMARY	The selected sub-channel index has been designated as being a primary sub-channel.
VE_USER_SUB_CHANNEL_INDEX_SECONDARY	The selected sub-channel index has been designated as being a secondary sub-channel.
VE_USER_SUB_CHANNEL_SELECT_COUNT	This element is used internally by VeriCall Edge and should not be used by the user application.

B.169 t_ve_user_callback_return_code_e

The **t_ve_user_callback_return_code_e** enumeration is used as the return code by all the **VeriCall Edge Data Manager Functions**, used to indicate the success or failure of the registered Data Manager Interface function. This enumeration is defined in **ve_callback.h** and is described below:

```
typedef enum{
  VE_USER_CALLBACK_SUCCESS   = 0x00,
  VE_USER_CALLBACK_FAIL      = 0x01
}t_ve_user_callback_return_code_e;
```

The enumeration elements are as follows:

Element	Definition
VE_USER_CALLBACK_SUCCESS	Indicates that the registered Data Manager Interface function returned successfully.
VE_USER_CALLBACK_FAIL	Indicates that the registered Data Manager Interface function returned with a failure.

B.170 t_ve_callback_transport_type_e

The **t_ve_callback_transport_type_e** enumeration is used by the **VE_RegisterPacketTransportOpen** function to indicate the packet-side transportation type to be used. This enumeration is defined in **ve_callback.h** and is described below:

```
typedef enum{
  VE_USER_CALL_BACK_TRANSPORT_TYPE_IP_V4,
  VE_USER_CALL_BACK_TRANSPORT_TYPE_IP_V6,
  VE_USER_CALL_BACK_TRANSPORT_TYPE_IP_FQDN
}t_ve_callback_transport_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_USER_CALL_BACK_TRANSPORT_TYPE_IP_V4	The transport type is an IPv4 type.
VE_USER_CALL_BACK_TRANSPORT_TYPE_IP_V6	Reserved for future use.
VE_USER_CALL_BACK_TRANSPORT_TYPE_IP_FQDN	The transport type is a fully qualified domain name type.

B.171 t_ve_callback_protocol_e

The **t_ve_callback_protocol_e** enumeration is used in calls to the function that was registered using the **VE_RegisterPacketTransportOpen** function to indicate the transportation protocol type used. This enumeration is defined in **ve_callback.h** and is described below:

```
typedef enum{
  VE_USER_CALL_BACK_TRANSPORT_PROTOCOL_UDP,
  VE_USER_CALL_BACK_TRANSPORT_PROTOCOL_TCP
}t_ve_callback_protocol_e;
```

The enumeration elements are as follows:

Element	Definition
VE_USER_CALL_BACK_TRANSPORT_PROTOCOL_UDP	The protocol used is UDP.
VE_USER_CALL_BACK_TRANSPORT_PROTOCOL_TCP	The protocol used is TCP.

B.172 t_ve_callback_direction_e

The **t_ve_callback_direction_e** enumeration is used in calls to the vendor written Data Manager functions that were registered with the VeriCall Edge system. These functions are described in Chapter 6 on Page 77. This enumerated type indicates whether the registered callback function is for the transmission or reception of packet-side and media-side data. This enumeration is defined in **ve_callback.h** and is described below:

```
typedef enum{
  VE_USER_CALL_BACK_TRANSPORT_DIRECTION_TX,
  VE_USER_CALL_BACK_TRANSPORT_DIRECTION_RX
}t_ve_callback_direction_e;
```

The enumeration elements are as follows:

Element	Definition
VE_USER_CALL_BACK_TRANSPORT_DIRECTION_TX	The registered transport data connection is for the transmission of packet data.
VE_USER_CALL_BACK_TRANSPORT_DIRECTION_RX	The registered transport data connection is for the reception of packet data.

B.173 t_ve_callback_audio_endian_e

The **t_ve_callback_audio_endian_e** enumeration is part of the **t_ve_callback_audio_data_channel_config_s** structure defined in Appendix B.22.1 on Page 296. The **t_ve_callback_audio_endian_e** enumeration determines the endian type to be used for the transmitted and received data through the registered Data Manager callback functions. The **t_ve_callback_audio_endian_e** enumeration is defined in **ve_callback.h** and is described below:

```
typedef enum{
  VE_USER_CALL_BACK_BIG_ENDIAN,
  VE_USER_CALL_BACK_LITTLE_ENDIAN
}t_ve_callback_audio_endian_e;
```

The enumeration elements are as follows:

Element	Definition
VE_USER_CALL_BACK_BIG_ENDIAN	The data is in Big Endian mode, i.e. MSB is encoded first.
VE_USER_CALL_BACK_LITTLE_ENDIAN	The data is in Little Endian mode, i.e. LSB is encoded first.

B.174 t_ve_callback_data_channel_media_format_e

The **t_ve_callback_data_channel_media_format_e** enumeration is used in calls to the function that was registered using the **VE_RegisterChannelDataDeviceConfigure** function. The **t_ve_callback_data_channel_media_format_e** enumeration determines the media data format to be expected by the VeriCall Edge system from the registered data manager callback functions. The **t_ve_callback_data_channel_media_format_e** enumeration is defined in **ve_callback.h** and is described below:

```
typedef enum{
  VE_USER_CALL_BACK_FORMAT_16BIT_PCM_LINEAR,
  VE_USER_CALL_BACK_FORMAT_G711U,
  VE_USER_CALL_BACK_FORMAT_G711A,
  VE_USER_CALL_BACK_FORMAT_VIDEO
}t_ve_callback_data_channel_media_format_e;
```

The enumeration elements are as follows:

Element	Definition
VE_USER_CALL_BACK_FORMAT_16BIT_PCM_LINEAR	The media format is 16-bit PCM linear.
VE_USER_CALL_BACK FORMAT_G711u	The media format is G.711µ.
VE_USER_CALL_BACK FORMAT_G711a	The media format is G.711a.
VE_USER_CALL_BACK_FORMAT _VIDEO	The media format is video.

B.175 t_ve_callback_video_color_format_e

The **t_ve_callback_video_color_format_e** enumeration is part of the **t_ve_callback_video_data_channel_config_s** structure defined in Appendix B.22.2 on Page 296. The **t_ve_callback_video_color_format_e** enumeration determines the supported image format. The enumeration is defined in **ve_callback.h** and is described below:

```
typedef enum{
  VE_VIDEO_CALLBACK_YUV420_COLOR_FORMAT = 0,
  VE_VIDEO_CALLBACK_YUV422_COLOR_FORMAT,
  VE_VIDEO_CALLBACK_RGB_16_BIT_COLOR_FORMAT,
  VE_VIDEO_CALLBACK_RGB_32_BIT_COLOR_FORMAT,
}t_ve_callback_video_color_format_e;
```

The enumeration elements are as follows:

Element	Definition
VE_VIDEO_CALLBACK_YUV420_COLOR_FORMAT	The supported image format is YU420.
VE_VIDEO_CALLBACK_YUV422_COLOR_FORMAT	The supported image format is YU422.
VE_VIDEO_CALLBACK_RGB_16_BIT_COLOR_FORMAT	The supported image format is 16-bit RGB.
VE_VIDEO_CALLBACK_RGB_32_BIT_COLOR_FORMAT	The supported image format is 32-bit RGB.

B.176 t_ve_callback_duplex_capability_e

The **t_ve_callback_duplex_capability_e** enumeration is part of the **t_ve_callback_audio_device_query_s** structure defined in Appendix B.36.1 on Page 427. The enumeration allows for the storage of particular audio devices duplex capabilities. The **t_ve_callback_duplex_capability_e** enumeration is defined in **ve_callback.h** and is described below:

```
typedef enum{
  VE_CALLBACK_AUDIO_DEVICE_HALF_DUPLEX = 0,
  VE_CALLBACK_AUDIO_DEVICE_FULL_DUPLEX
}t_ve_callback_duplex_capability_e;
```

The enumeration elements are as follows:

Element	Definition
VE_CALLBACK_AUDIO_DEVICE_HALF_DUPLEX	The queried audio device operates in half-duplex mode.
VE_CALLBACK_AUDIO_DEVICE_FULL_DUPLEX	The queried audio device operates in full-duplex mode.

B.177 t_ve_callback_transfermode_e

The **t_ve_callback_transfermode_e** enumeration is part of the **t_ve_callback_video_capture_device_query_s** structure defined in Appendix B.36.3 on Page 429. The enumeration allows for the storage of particular video capture devices method of data availability indication. The **t_ve_callback_transfermode_e** enumeration is defined in **ve_callback.h** and is described below:

```
typedef enum{
  VE_CALLBACK_TRANSFER_MODE_POLLED = 0,
  VE_CALLBACK_TRANSFER_MODE_INTERRUPT
}t_ve_callback_transfermode_e;
```

The enumeration elements are as follows:

Element	Definition
VE_CALLBACK_TRANSFER_MODE_POLLED	The video capture device will be polled for available data.
VE_CALLBACK_TRANSFER_MODE_INTERRUPT	The video capture device will indicate data availability using interrupts.

B.178 t_ve_callback_video_frame_buffer_e

The **t_ve_callback_video_frame_buffer_e** enumeration is part of the **t_ve_callback_video_display_device_query_s** structure defined in Appendix B.36.2 on Page 428. The enumeration allows for the type of the video frame buffer to be defined. The **t_ve_callback_video_frame_buffer_e** enumeration is defined in **ve_callback.h** and is described below:

```
typedef enum{
  VE_CALLBACK_VIDEO_FRAME_BUFFER_INVALID = 0,
  VE_CALLBACK_VIDEO_FRAME_BUFFER_STATIC,
  VE_CALLBACK_VIDEO_FRAME_BUFFER_VIRTUAL
}t_ve_callback_video_frame_buffer_e;
```

The enumeration elements are as follows:

Element	Definition
VE_CALLBACK_VIDEO_FRAME_BUFFER_INVALID	This is an invalid video frame buffer selection.
VE_CALLBACK_VIDEO_FRAME_BUFFER_STATIC	The frame buffer is used for a primary or base graphics window.
VE_CALLBACK_VIDEO_FRAME_BUFFER_VIRTUAL	The frame buffer is used for a secondary graphics window, often referred to as the graphics plane.

B.179 t_ve_stats_type_e

The **t_ve_stats_type_e** enumeration is used by the **VE_GetStatistics** function call to specify the statistics type to access. This enumeration is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_STATISTICS_TYPE_SYSTEM_CALLS,
  VE_STATISTICS_TYPE_CHANNEL_VIDEO_ENCODER,
  VE_STATISTICS_TYPE_CHANNEL_VIDEO_DECODER,
  VE_STATISTICS_TYPE_CHANNEL_RTCP,
  VE_STATISTICS_TYPE_CHANNEL_RTCP_PEER,
  VE_STATISTICS_TYPE_CHANNEL_AV_SYNC,
  VE_NUM_STATISTICS_TYPES
}t_ve_stats_type_e;
```

The enumeration elements of **t_ve_stats_type_e** are as follows:

Element	Definition
VE_STATISTICS_TYPE_SYSTEM_CALLS	The requested statistics are those for the number VeriCall Edge system calls made.
VE_STATISTICS_TYPE_CHANNEL_VIDEO_ENCODER	The requested statistics are those for a channels video encoder.
VE_STATISTICS_TYPE_CHANNEL_VIDEO_DECODER	The requested statistics are those for a channels video decoder.
VE_STATISTICS_TYPE_CHANNEL_RTCP	The requested statistics are for a channels RTCP.
VE_STATISTICS_TYPE_CHANNEL_RTCP_PEER	The requested statistics are for a channels RTCP peer.
VE_STATISTICS_TYPE_CHANNEL_AV_SYNC	The requested statistics are for a channels audio/video synchronization statistics.
VE_NUM_STATISTICS_TYPES	The number of statistic types contained in this enumeration. This is used internally by the VeriCall Edge system and should not be used by the user application.

B.180 t_ve_stat_av_sync_correlate_e

The **t_ve_stat_av_sync_correlate_e** enumeration is used within the **t_ve_stat_av_sync_s** structure, described in Appendix B.38.2.1 on Page 433. This enumeration specifies whether the audio to visual synchronization statistics are correlated. The **t_ve_stat_av_sync_correlate_e** enumeration is defined in **ve_defs.h** and is described below:

```
typedef enum{
  VE_STAT_AV_SYNC_CORRELATE_TRUE,
  VE_STAT_AV_SYNC_CORRELATE_FALSE,
}t_ve_stat_av_sync_correlate_e;
```

The enumeration elements of **t_ve_stat_av_sync_correlate_e** are as follows:

Element	Definition
VE_STAT_AV_SYNC_CORRELATE_TRUE	The audio to video synchronization statistics are correlated.
VE_STAT_AV_SYNC_CORRELATE_FALSE	The audio to video synchronization statistics are not correlated.

B.181 t_ve_sip_response_code_e

The **t_ve_sip_response_code_e** enumeration is returned to calls made to **VE_SIPSessionResponse**. The enumeration provides the standard SIP response codes, allowing for easy decoding of the SIP response code. The **t_ve_sip_response_code_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_RESPONSE_UNKNOWN=0,

  /* 1xx: Provisional */
  VE_SIP_RESPONSE_100_TRYING                          = 100,
  VE_SIP_RESPONSE_180_RINGING                         = 180,
  VE_SIP_RESPONSE_181_CALL_IS_BEING_FORWARDED         = 181,
  VE_SIP_RESPONSE_182_QUEUED                          = 182,
  VE_SIP_RESPONSE_183_SESSION_PROGRESS                = 183,

  /* 2xx: Success */
  VE_SIP_RESPONSE_200_OK                              = 200,
  VE_SIP_RESPONSE_202_ACCEPTED                        = 202,

  /* 3xx: Redirection */
  VE_SIP_RESPONSE_300_MULTIPLE_CHOICES                = 300,
  VE_SIP_RESPONSE_301_MOVED_PERMANENTLY               = 301,
  VE_SIP_RESPONSE_302_MOVED_TEMPORARILY               = 302,
  VE_SIP_RESPONSE_305_USE_PROXY                       = 305,
  VE_SIP_RESPONSE_380_ALTERNATIVE_SERVICE             = 380,

  /* 4xx: Client Error */
  VE_SIP_RESPONSE_400_BAD_REQUEST                     = 400,
  VE_SIP_RESPONSE_401_UNAUTHORIZED                    = 401,
  VE_SIP_RESPONSE_402_PAYMENT_REQUIRED                = 402,
  VE_SIP_RESPONSE_403_FORBIDDEN                       = 403,
  VE_SIP_RESPONSE_404_NOT_FOUND                       = 404,
  VE_SIP_RESPONSE_405_METHOD_NOT_ALLOWED              = 405,
  VE_SIP_RESPONSE_406_NOT_ACCEPTABLE                  = 406,
  VE_SIP_RESPONSE_407_PROXY_AUTHENTICATION_REQUIRED   = 407,
  VE_SIP_RESPONSE_408_REQUEST_TIMEOUT                 = 408,
  VE_SIP_RESPONSE_410_GONE                            = 410,
  VE_SIP_RESPONSE_413_REQUEST_ENTITY_TOO_LARGE        = 413,
  VE_SIP_RESPONSE_414_REQUEST_URI_TOO_LONG            = 414,
  VE_SIP_RESPONSE_415_UNSUPPORTED_MEDIA_TYPE          = 415,
  VE_SIP_RESPONSE_416_UNSUPPORTED_URI_SCHEME          = 416,
  VE_SIP_RESPONSE_420_BAD_EXTENSION                   = 420,
  VE_SIP_RESPONSE_421_EXTENSION_REQUIRED              = 421,
  VE_SIP_RESPONSE_423_INTERVAL_TOO_BRIEF              = 423,
  VE_SIP_RESPONSE_429_PROVIDE_REFERRER_IDENTITY       = 429,
  VE_SIP_RESPONSE_480_TEMPORARILY_UNAVAILABLE         = 480,
  VE_SIP_RESPONSE_481_CALL_TRANSACTION_DOES_NOT_EXIST = 481,
```

Continued >>

```
    VE_SIP_RESPONSE_482_LOOP_DETECTED                    = 482,
    VE_SIP_RESPONSE_483_TOO_MANY_HOPS                    = 483,
    VE_SIP_RESPONSE_484_ADDRESS_INCOMPLETE               = 484,
    VE_SIP_RESPONSE_485_AMBIGUOUS                        = 485,
    VE_SIP_RESPONSE_486_BUSY_HERE                        = 486,
    VE_SIP_RESPONSE_487_REQUEST_TERMINATED               = 487,
    VE_SIP_RESPONSE_488_NOT_ACCEPTABLE_HERE              = 488,
    VE_SIP_RESPONSE_489_BAD_EVENT                        = 489,
    VE_SIP_RESPONSE_491_REQUEST_PENDING                  = 491,
    VE_SIP_RESPONSE_493_UNDECIPHERABLE                   = 493,

    /* 5xx: Server Error */
    VE_SIP_RESPONSE_500_SERVER_INTERNAL_ERROR            = 500,
    VE_SIP_RESPONSE_501_NOT_IMPLEMENTED                  = 501,
    VE_SIP_RESPONSE_502_BAD_GATEWAY                      = 502,
    VE_SIP_RESPONSE_503_SERVICE_UNAVAILABLE              = 503,
    VE_SIP_RESPONSE_504_SERVER_TIME_OUT                  = 504,
    VE_SIP_RESPONSE_505_VERSION_NOT_SUPPORTED            = 505,
    VE_SIP_RESPONSE_513_MESSAGE_TOO_LARGE                = 513,

    /* 6xx: Global Failure */
    VE_SIP_RESPONSE_600_BUSY_EVERYWHERE                  = 600,
    VE_SIP_RESPONSE_603_DECLINE                          = 603,
    VE_SIP_RESPONSE_604_DOES_NOT_EXIST_ANYWHERE          = 604,
    VE_SIP_RESPONSE_606_NOT_ACCEPTABLE                   = 606
}t_ve_sip_response_code_e;
```

The SIP response codes are defined within RFC3261, RFC3265 and RFC3892.

B.182 t_ve_sip_symmetric_mode_default_configuration_e

The **t_ve_sip_symmetric_mode_default_configuration_e** enumeration is used within the **t_ve_sip_default_configuration_s** structure, described in Appendix B.17 on Page 284. The enumeration determines the default configuration mode of operation of the SIP module. The **t_ve_sip_symmetric_mode_default_configuration_e** enumeration is defined in **ve_sip_configuration.h** and is described below:

```
typedef enum{
  VE_SIP_SYMMETRIC_MODE_DEFAULT_CONFIGURATION_ASYMMETRIC,
  VE_SIP_SYMMETRIC_MODE_DEFAULT_CONFIGURATION_SYMMETRIC
}t_ve_sip_symmetric_mode_default_configuration_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_SYMMETRIC_MODE_DEFAULT_ CONFIGURATION_ASYMMETRIC	The default SIP configuration is asymmetric and SIP messages will be sent from a random port selected by the operating system.
VE_SIP_SYMMETRIC_MODE_DEFAULT_ CONFIGURATION_SYMMETRIC	The default SIP configuration is symmetric and SIP messages will be sent from port 5060.

B.183 t_ve_sip_state_flag_default_configuration_e

The **t_ve_sip_state_flag_default_configuration_e** enumeration is used within the following SIP structures:

- **t_ve_sip_default_configuration_s** as described in Appendix B.17 on Page 284.
- **t_ve_sip_stack_liberalisation_s** as described in Appendix B.40 on Page 443.
- **t_ve_sip_stack_feature_support_s** as described in Appendix B.41 on Page 443.
- **t_ve_sip_routing_s** as described in Appendix B.42 on Page 445.

The enumeration provides a state flag for specific default configuration elements to be enabled or disabled. The **t_ve_sip_state_flag_default_configuration_e** enumeration is defined in **ve_sip_configuration.h** and is described below:

```
typedef enum{
  VE_SIP_STATE_FLAG_DEFAULT_CONFIGURATION_ENABLED,
  VE_SIP_STATE_FLAG_DEFAULT_CONFIGURATION_DISABLED
}t_ve_sip_state_flag_default_configuration_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_STATE_FLAG_DEFAULT_ CONFIGURATION_ENABLED	The specified default configuration option is enabled.
VE_SIP_STATE_FLAG_DEFAULT_ CONFIGURATION_DISABLED	The specified default configuration option is disabled.

B.184 t_ve_sip_deregister_contacts_flag_e

The **t_ve_sip_deregister_contacts_flag_e** enumeration is used as an argument in **VE_SIPDeregisterUser** calls. The enumeration determines the level of contacts to be deregistered. The **t_ve_sip_deregister_contacts_flag_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_DEREGISTER_LOCAL_CONTACTS = 0,
  VE_SIP_DEREGISTER_ALL_CONTACTS
}t_ve_sip_deregister_contacts_flag_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_DEREGISTER_LOCAL_CONTACTS	Deregister only local contacts.
VE_SIP_DEREGISTER_ALL_CONTACTS	Deregister all contacts.

B.185 t_ve_sip_require_prack_e

The **t_ve_sip_require_prack_e** enumeration is used within the **t_ve_sip_tx_session_response_data_s** structure, described in Appendix B.44 on Page 448, and the **t_ve_remote_user_uri_s** structure, described in Appendix B.51 on Page 466. The enumeration is used to identify the need for PRACK. The **t_ve_sip_require_prack_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_REQUIRE_PRACK_NO=0,
  VE_SIP_REQUIRE_PRACK_YES
}t_ve_sip_require_prack_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_REQUIRE_PRACK_NO	PRACK is not required or is not present.
VE_SIP_REQUIRE_PRACK_YES	PRACK is required or is present.

Note that the PRACK header will only be added if the destination supports the feature.

B.186 t_ve_sip_date_state_e

The **t_ve_sip_date_state_e** enumeration is used within the **t_ve_sip_date_s** structure, described in Appendix B.43 on Page 447. The enumeration is used to specify whether use of the SIP date header is valid. The **t_ve_sip_date_state_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_DATE_STATE_INACTIVE=0,
  VE_SIP_DATE_STATE_ACTIVE
}t_ve_sip_date_state_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_DATE_STATE_INACTIVE	The SIP date header is not used.
VE_SIP_DATE_STATE_ACTIVE	The SIP date header is used.

B.187 t_ve_sip_tx_session_response_content_e

The **t_ve_sip_tx_session_response_content_e** enumeration is used within the **t_ve_sip_tx_session_response_data_s** structure, described in Appendix B.44 on Page 448. The enumeration is used to specify the SIP transmit session response content type. The **t_ve_sip_tx_session_response_content_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_TX_CONTENT_AUTOMATIC=0,
  VE_SIP_TX_CONTENT_INCLUDE_SDP,
  VE_SIP_TX_CONTENT_INCLUDE_SDP_FROM_STREAM_CONFIG,
  VE_SIP_TX_CONTENT_INCLUDE_CONTENT_DATA
}t_ve_sip_tx_session_response_content_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_TX_CONTENT_AUTOMATIC	The SIP stack will automatically add the content as needed. This is the default behavior
VE_SIP_TX_CONTENT_INCLUDE_SDP	SDP will be forced to be included in the response. This only applies to responses to a SIP INVITE.
VE_SIP_TX_CONTENT_INCLUDE_SDP _FROM_STREAM_CONFIG	SDP is forced to be included by using the stream configuration data provided by the user application. This only applies to responses to a SIP INVITE.
VE_SIP_TX_CONTENT_INCLUDE_CONTENT_DATA	Include the content data in the response.

B.188 t_ve_sip_authentication_required_e

The **t_ve_sip_authentication_required_e** enumeration is used within the **t_ve_sip_authentication_data_s** structure, described in Appendix B.45.1 on Page 450. The enumeration is a flag that indicates whether SIP authentication is to be used or not. The **t_ve_sip_authentication_required_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_AUTHENTICATION_REQUIRED_NO=0,
  VE_SIP_AUTHENTICATION_REQUIRED_YES
}t_ve_sip_authentication_required_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_AUTHENTICATION_REQUIRED_NO	SIP authentication is not required.
VE_SIP_AUTHENTICATION_REQUIRED_YES	SIP authentication is required.

B.189 t_ve_sip_rx_session_response_content_e

The **t_ve_sip_rx_session_response_content_e** enumeration is used within the **t_ve_sip_rx_session_response_data_s** structure, described in Appendix B.45 on Page 449. The enumeration specifies the received SIP response content type. The **t_ve_sip_rx_session_response_content_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_RX_CONTENT_NONE=0,
  VE_SIP_RX_CONTENT_SDP,
  VE_SIP_RX_CONTENT_UNKNOWN
}t_ve_sip_rx_session_response_content_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_RX_CONTENT_NONE	There is no content.
VE_SIP_RX_CONTENT_SDP	The content is of Session Description Protocol (SDP) type.
VE_SIP_RX_CONTENT_UNKNOWN	The content type is not recognized but it has been passed up to the user application.

B.190 t_ve_sip_call_hold_method_e

The **t_ve_sip_call_hold_method_e** enumeration is used as an argument to the **VE_SIPSessionHoldOn** and **VE_SIPSessionHoldOff** calls. The enumeration specifies the SIP call hold method. The **t_ve_sip_call_hold_method_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_CALL_HOLD_METHOD_INVALID=0,
  VE_SIP_CALL_HOLD_METHOD_RFC_2543,
  VE_SIP_CALL_HOLD_METHOD_RFC_3264,
  VE_NUM_SIP_CALL_HOLD_METHODS
}t_ve_sip_call_hold_method_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_CALL_HOLD_METHOD_INVALID	Use internally by the VeriCall Edge system, should not be used by the user application.
VE_SIP_CALL_HOLD_METHOD_RFC_2543	Perform call hold as per RFC2543; set the SDP connection IP address to zero.
VE_SIP_CALL_HOLD_METHOD_RFC_3264	Perform call hold as per RFC3264; use send only or inactive in the SDP.
VE_NUM_SIP_CALL_HOLD_METHODS	Use internally by the VeriCall Edge system, should not be used by the user application.

B.191 t_ve_sip_call_hold_state_e

The **t_ve_sip_call_hold_state_e** enumeration is used within the **t_ve_user_msg_sip_reinvite_response_rx_s** structure, described in Appendix B.33.13 on Page 370. The enumeration specifies the current SIP call hold state. The **t_ve_sip_call_hold_state_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_CALL_HOLD_STATE_OFF=0,
  VE_SIP_CALL_HOLD_STATE_ON
}t_ve_sip_call_hold_state_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_CALL_HOLD_STATE_OFF	The call is currently not held.
VE_SIP_CALL_HOLD_STATE_ON	The call is currently held.

B.192 t_ve_sip_presence_xml_pidf_status_basic_type_e

The **t_ve_sip_presence_xml_pidf_status_basic_type_e** enumeration is used within the **t_ve_sip_presence_xml_pidf_status_s** structure as described in Appendix B.46.1.2 on Page 455. The enumeration specifies the SIP basic PIDF status. The **t_ve_sip_presence_xml_pidf_status_basic_type_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_PRESENCE_XML_PIDF_STATUS_INVALID=0,
  VE_SIP_PRESENCE_XML_PIDF_STATUS_BASIC_OPEN,
  VE_SIP_PRESENCE_XML_PIDF_STATUS_BASIC_CLOSED,
}t_ve_sip_presence_xml_pidf_status_basic_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_PRESENCE_XML_PIDF_STATUS_INVALID	Use internally by the VeriCall Edge system, should not be used by the user application.
VE_SIP_PRESENCE_XML_PIDF_STATUS_BASIC_OPEN	The basic PIDF status is open.
VE_SIP_PRESENCE_XML_PIDF_STATUS_BASIC_CLOSED	The basic PIDF status is closed.

B.193 t_ve_sip_request_type_e

The **t_ve_sip_request_type_e** enumeration is used within the **t_ve_user_msg_sip_operation_failed_s** structure, as described in Appendix B.33.26 on Page 383. The enumerated request type corresponds to the SIP function which made the request. The **t_ve_sip_request_type_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_REQUEST_TYPE_INVALID=0,
  VE_SIP_REQUEST_TYPE_VE_SIPSESSIONSTART,
  VE_SIP_REQUEST_TYPE_VE_SIPSESSIONSTART_WITHCONFIG,
  VE_SIP_REQUEST_TYPE_VE_SIPSESSIONSTOP,
  VE_SIP_REQUEST_TYPE_VE_SIPSESSIONRESPONSE,
  VE_SIP_REQUEST_TYPE_VE_SIPREGISTERUSER,
  VE_SIP_REQUEST_TYPE_VE_SIPDEREGISTERUSER,
  VE_SIP_REQUEST_TYPE_VE_SIPUPDATEREGISTRATION,
  VE_SIP_REQUEST_TYPE_VE_SIPQUERYREGISTRATION,
  VE_SIP_REQUEST_TYPE_VE_SIPSESSIONHOLDON,
  VE_SIP_REQUEST_TYPE_VE_SIPSESSIONHOLDOFF,
  VE_SIP_REQUEST_TYPE_VE_SIPSESSIONTRANSFERCONSULT,
  VE_SIP_REQUEST_TYPE_VE_SIPSESSIONTRANSFERDONTCONSULT,
  VE_SIP_REQUEST_TYPE_VE_SIPSESSIONTRANSFER,
  VE_SIP_REQUEST_TYPE_VE_SIPSESSIONUPDATE,
  VE_SIP_REQUEST_TYPE_VE_SIPSESSIONREINVITE,
  VE_SIP_REQUEST_TYPE_VE_SIPSENDMESSAGE,
  VE_SIP_REQUEST_TYPE_VE_SIPSENDOPTIONS,
  VE_SIP_REQUEST_TYPE_VE_SIPSENDINFO,
  VE_SIP_REQUEST_TYPE_VE_SIPSENDSERVICE,
  VE_SIP_REQUEST_TYPE_VE_SIPSTARTPRESENCESUBSCRIPTION,
  VE_SIP_REQUEST_TYPE_VE_SIPSTOPPRESENCESUBSCRIPTION,
  VE_SIP_REQUEST_TYPE_VE_SIPSTARTMWSUBSCRIPTION,
  VE_SIP_REQUEST_TYPE_VE_SIPSTOPMWSUBSCRIPTION,
  VE_SIP_REQUEST_TYPE_VE_SIPINTERROGATEMWSUBSCRIPTION,
  VE_SIP_REQUEST_TYPE_VE_SIPSTARTDIALOGSUBSCRIPTION,
   VE_SIP_REQUEST_TYPE_VE_SIPSTOPDIALOGSUBSCRIPTION,
   VE_SIP_REQUEST_TYPE_VE_SIPINTERROGATEDIALOGSUBSCRIPTION
}t_ve_sip_request_type_e;
```

Continued >>

The enumeration elements are as follows:

Element	Definition
VE_SIP_REQUEST_TYPE_INVALID	Use internally by the VeriCall Edge system, should not be used by the user application.
VE_SIP_REQUEST_TYPE_VE_ SIPSESSIONSTART	The **VE_SIPSessionStart** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSESSIONSTART_WITHCONFIG	The **VE_SIPSessionStart_WithConfig** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSESSIONSTOP	The **VE_SIPSessionStop** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSESSIONRESPONSE	The **VE_SIPSessionResponse** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPREGISTERUSER	The **VE_SIPRegisterUser** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPDEREGISTERUSER	The **VE_SIPRegisterUser** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPUPDATEREGISTRATION	The **VE_SIPUpdateRegistration** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPQUERYREGISTRATION	The **VE_SIPQueryRegistration** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSESSIONHOLDON	The **VE_SIPSessionHoldOn** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSESSIONHOLDOFF	The **VE_SIPSessionHoldOff** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSESSIONTRANSFERCONSULT	The **VE_SIPSessionTransferConsult** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSESSIONTRANSFERDONTCONSULT	The **VE_SIPSessionTransferDontConsult** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSESSIONTRANSFER	The **VE_SIPSessionTransfer** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSESSIONUPDATE	The **VE_SIPSessionUpdate** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSESSIONREINVITE	The **VE_SIPSessionReinvite** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSENDMESSAGE	The **VE_SIPSendMessage** function call failed.
VE_SIP_REQUEST_TYPE_VE _SIPSENDOPTIONS	The **VE_SIPSendOptions** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSENDINFO	The **VE_SIPSendInfo** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSENDSERVICE	The **VE_SIPSendService** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSTARTPRESENCESUBSCRIPTION	The **VE_SIPStartPresenceSubscription** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSTOPPRESENCESUBSCRIPTION	The **VE_SIPStopPresenceSubscription** function call failed.

Element	Definition
VE_SIP_REQUEST_TYPE_VE_ SIPSTARTMWSUBSCRIPTION	The **VE_SIPStartMessageWaitingSubscription** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSTOPMWSUBSCRIPTION	The **VE_SIPStopMessageWaitingSubscription** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPINTERROGATEMWSUBSCRIPTION	The **VE_SIPInterrogateMessageWaiting** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSTARTDIALOGSUBSCRIPTION	The **VE_SIPStartDialogSubscription** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPSTOPDIALOGSUBSCRIPTION	The **VE_SIPStopDialogSubscription** function call failed.
VE_SIP_REQUEST_TYPE_VE_ SIPINTERROGATEDIALOGSUBSCRIPTION	The **VE_SIPInterrogateDialogSubscription** function call failed.

B.194 t_ve_sip_subscribe_state_e

The **t_ve_sip_subscribe_state_e** enumeration is used within the following structures:

- **t_ve_user_msg_sip_presence_subscribe_rx_s** described in Appendix B.33.27 on Page 384.
- **t_ve_user_msg_sip_presence_notify_rx_s** described in Appendix B.33.28 on Page 385.
- **t_ve_user_msg_sip_mw_notify_rx_s** described in Appendix B.33.30 on Page 387.

The enumeration specifies the current SIP subscribe state. The **t_ve_sip_subscribe_state_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_SUBSCRIBE_STATE_INVALID=0,
  VE_SIP_SUBSCRIBE_STATE_ACTIVE,
  VE_SIP_SUBSCRIBE_STATE_PENDING,
  VE_SIP_SUBSCRIBE_STATE_TERMINATED
}t_ve_sip_subscribe_state_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_SUBSCRIBE_STATE_INVALID	Use internally by the VeriCall Edge system, should not be used by the user application.
VE_SIP_SUBSCRIBE_STATE_ACTIVE	The SIP subscribe state is active.
VE_SIP_SUBSCRIBE_STATE_PENDING	The SIP subscribe state is pending.
VE_SIP_SUBSCRIBE_STATE_TERMINATED	The SIP subscribe state is terminated.

B.195 t_ve_sip_presence_type_e

The **t_ve_sip_presence_type_e** enumeration is used within the **t_ve_sip_presence_data_s** structure, as described in Appendix B.46 on Page 452. The enumeration specifies the current SIP presence type. The **t_ve_sip_presence_type_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_PRESENCE_TYPE_INVALID=0,
  VE_SIP_PRESENCE_TYPE_XML_PIDF
}t_ve_sip_presence_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_PRESENCE_TYPE_INVALID	Use internally by the VeriCall Edge system, should not be used by the user application.
VE_SIP_PRESENCE_TYPE_XML_PIDF	The SIP presence type is PIDF.

B.196 t_ve_sip_mw_start_flag_e

The **t_ve_sip_mw_start_flag_e** enumeration is used as an argument in calls to **VE_SIPStartMessageWaitingSubscription**. This enumeration specifies the Message Waiting (MW) start type to be used. The **t_ve_sip_mw_start_flag_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_MESSAGES_WAITING_START_INVALID=0,
  VE_SIP_MESSAGES_WAITING_START_RFC3842,
  VE_SIP_MESSAGES_WAITING_START_ALLOW_UNSOLICITED_NOTIFY,
  VE_NUM_SIP_MESSAGES_WAITING_START_FLAG
}t_ve_sip_mw_start_flag_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_MESSAGES_WAITING_START_INVALID	Use internally by the VeriCall Edge system, should not be used by the user application.
VE_SIP_MESSAGES_WAITING_START_RFC3842	Specify the MW starting method as per RFC3842. In this mode of operation the user application should expect to receive a confirmation event indicating the success or failure of the subscription attempt. If successful the user application will receive a `NOTIFY` event containing the current status of the given mailbox.
VE_SIP_MESSAGES_WAITING_START_ALLOW_ UNSOLICITED_NOTIFY	Instructs the MW starting method to ignore RFC3842, thus allowing unsolicited messages-summary `NOTIFY` messages. The user application will not receive any success or failure events as no actual SIP subscription was made. The user application will also not receive an immediate `NOTIFY` event containing the current status of the given mailbox. Since no subscription was made, **VE_SIPInterrogateMessageWaiting** will do nothing.
VE_NUM_SIP_MESSAGES_WAITING_START_FLAG	Use internally by the VeriCall Edge system, should not be used by the user application.

The ability to select the start method VE_SIP_MESSAGES_WAITING_START_ALLOW_ UNSOLICITED_NOTIFY has been added to allow operation with some SIP servers which send the mailbox `NOTIFY` message after a registration is made. If the user application is to operate with these types of servers, Trinity Convergence recommends that the VE_SIPStartMessageWaitingSubscription is called prior to VE_SIPRegisterUser, otherwise the user application may not receive any mailbox `NOTIFY` messages until the next re-register message (default is 45minutes).

B.197 t_ve_sip_mw_indicatorstatus_e

The **t_ve_sip_mw_indicatorstatus_e** enumeration is used within the **t_ve_sip_mw_data_s** structure, as described in Appendix B.47 on Page 457. This enumeration specifies whether there is a message waiting. The **t_ve_sip_mw_indicatorstatus_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_MESSAGES_WAITING_INVALID=0,
  VE_SIP_MESSAGES_WAITING_YES,
  VE_SIP_MESSAGES_WAITING_NO,
  VE_NUM_SIP_MESSAGES_WAITING
}t_ve_sip_mw_indicatorstatus_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_MESSAGES_WAITING_INVALID	Use internally by the VeriCall Edge system, should not be used by the user application.
VE_SIP_MESSAGES_WAITING_YES	The mailbox server is instructing the user application to set its Message Waiting Indicator.
VE_SIP_MESSAGES_WAITING_NO	The mailbox server is instructing the user application to clear its Message Waiting Indicator.
VE_NUM_SIP_MESSAGES_WAITING	Use internally by the VeriCall Edge system, should not be used by the user application.

It is up to the discretion of the mailbox server to set and clear the indicator, most probably the server will set `VE_SIP_MESSAGES_WAITING_YES` if there are any new messages in the mailbox.

B.198 t_ve_sip_mw_message_context_e

The **t_ve_sip_mw_message_context_e** enumeration is used within the **t_ve_sip_mw_msg_summary_data_s** structure, as described in Appendix B.47.1 on Page 458. This enumeration specifies the type of message that is available. The **t_ve_sip_mw_message_context_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_MW_CONTEXT_NONE = 0,
  VE_SIP_MW_CONTEXT_VOICE_MESSAGE,
  VE_SIP_MW_CONTEXT_FAX_MESSAGE,
  VE_SIP_MW_CONTEXT_PAGER_MESSAGE,
  VE_SIP_MW_CONTEXT_MULTIMEDIA_MESSAGE,
  VE_SIP_MW_CONTEXT_TEXT_MESSAGE
}t_ve_sip_mw_message_context_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_MW_CONTEXT_NONE	Use internally by the VeriCall Edge system, should not be used by the user application.
VE_SIP_MW_CONTEXT_VOICE_MESSAGE	The message available is a voice type message.
VE_SIP_MW_CONTEXT_FAX_MESSAGE	The message available is a FAX type message.
VE_SIP_MW_CONTEXT_PAGER_MESSAGE	The message available is a pager type message.
VE_SIP_MW_CONTEXT_MULTIMEDIA_MESSAGE	The message available is a multimedia type message.
VE_SIP_MW_CONTEXT_TEXT_MESSAGE	The message available is a text type message.

B.199 t_ve_sip_dialog_field_state_e

The **t_ve_sip_dialog_field_state_e** enumeration is used within the following structures.

- **t_ve_sip_dialog_state_s** structure, as described in Appendix B.48.1.2 on Page 461.
- **t_ve_sip_dialog_direction_s** structure, as described in Appendix B.48.1.3 on Page 462.
- **t_ve_sip_dialog_duration_s** structure, as described in Appendix B.48.1.4 on Page 462.

The enumeration defines the possible states of a SIP dialog field. The **t_ve_sip_dialog_field_state_e** enumeration is defined in **ve_sip_defs.h** and described below:

```
typedef enum{
  VE_SIP_DIALOG_FIELD_STATE_INACTIVE = 0,
  VE_SIP_DIALOG_FIELD_STATE_ACTIVE
}t_ve_sip_dialog_field_state_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_DIALOG_FIELD_STATE_INACTIVE	Field inactive and data in the structure is not valid.
VE_SIP_DIALOG_FIELD_STATE_ACTIVE	Field active and data in the structure is valid.

B.200 t_ve_sip_dialog_state_e

The **t_ve_sip_dialog_state_e** enumeration is used within the **t_ve_sip_dialog_state_s** structure, as described in Appendix B.48.1.2 on Page 461. The enumeration defines the possible states of a SIP dialog. The **t_ve_sip_dialog_state_e** enumeration is defined in **ve_sip_defs.h** and described below:

```
typedef enum{
  VE_SIP_DIALOG_STATE_INVALID = 0,
  VE_SIP_DIALOG_STATE_TRYING,
  VE_SIP_DIALOG_STATE_PROCEEDING,
  VE_SIP_DIALOG_STATE_EARLY,
  VE_SIP_DIALOG_STATE_CONFIRMED,
  VE_SIP_DIALOG_STATE_TERMINATED,
  VE_SIP_NUM_DIALOG_STATES
}t_ve_sip_dialog_state_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_DIALOG_STATE_INVALID	Invalid SIP dialog state.
VE_SIP_DIALOG_STATE_TRYING	SIP dialog state is Trying. An INVITE has been sent/received but no response has been received/sent.
VE_SIP_DIALOG_STATE_PROCEEDING	SIP dialog state is Proceeding. An INVITE has been sent/received and a 1xx response without a tag has been received/sent.
VE_SIP_DIALOG_STATE_EARLY	SIP dialog state is Early. An INVITE has been sent/received and a 1xx response with a tag has been received/sent.
VE_SIP_DIALOG_STATE_CONFIRMED	SIP dialog state is Confirmed. An INVITE has been sent/received and a 2xx response has been received/sent.
VE_SIP_DIALOG_STATE_TERMINATED	SIP dialog state is Terminated. The reason for the termination is provided by the enumerated type **t_ve_sip_dialog_state_event_e**, as described in Appendix B.201 on Page 621.
VE_SIP_NUM_DIALOG_STATES	The number of elements in this enumeration.

B.201 t_ve_sip_dialog_state_event_e

The **t_ve_sip_dialog_state_event_e** enumeration is used within the **t_ve_sip_dialog_state_s** structure, as described in Appendix B.48.1.2 on Page 461. The enumeration defines the possible events that can cause a change to the state of a SIP dialog. The **t_ve_sip_dialog_state_event_e** enumeration is defined in **ve_sip_defs.h** and described below:

```
typedef enum{
  VE_SIP_DIALOG_STATE_EVENT_INVALID = 0,
  VE_SIP_DIALOG_STATE_EVENT_CANCELLED,
  VE_SIP_DIALOG_STATE_EVENT_REJECTED,
  VE_SIP_DIALOG_STATE_EVENT_REPLACED,
  VE_SIP_DIALOG_STATE_EVENT_LOCAL_BYE,
  VE_SIP_DIALOG_STATE_EVENT_REMOTE_BYE,
  VE_SIP_DIALOG_STATE_EVENT_ERROR,
  VE_SIP_DIALOG_STATE_EVENT_TIMEOUT,
  VE_SIP_NUM_DIALOG_STATE_EVENTS
}t_ve_sip_dialog_state_event_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_DIALOG_STATE_EVENT_INVALID	Invalid SIP dialog state event.
VE_SIP_DIALOG_STATE_EVENT_CANCELLED	Call cancelled.
VE_SIP_DIALOG_STATE_EVENT_REJECTED	Call rejected.
VE_SIP_DIALOG_STATE_EVENT_REPLACED	Dialog replaced by new dialog.
VE_SIP_DIALOG_STATE_EVENT_LOCAL_BYE	Local Bye. Initiator terminated the call
VE_SIP_DIALOG_STATE_EVENT_REMOTE_BYE	Remote Bye. Recipient terminated the call.
VE_SIP_DIALOG_STATE_EVENT_ERROR	An error terminated the call.
VE_SIP_DIALOG_STATE_EVENT_TIMEOUT	A timeout terminated the call.
VE_SIP_NUM_DIALOG_STATE_EVENTS	The number of elements in this enumeration.

B.202 t_ve_sip_dialog_direction_e

The **t_ve_sip_dialog_direction_e** enumeration is used within the **t_ve_sip_dialog_direction_s** structure, as described in Appendix B.48.1.3 on Page 462. The enumeration defines the possible directions of a SIP dialog. The **t_ve_sip_dialog_direction_e** enumeration is defined in **ve_sip_defs.h** and described below:

```
typedef enum{
  VE_SIP_DIALOG_DIRECTION_INVALID = 0,
  VE_SIP_DIALOG_DIRECTION_INITIATOR,
  VE_SIP_DIALOG_DIRECTION_RECIPIENT,
  VE_SIP_NUM_DIALOG_DIRECTIONS
}t_ve_sip_dialog_direction_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_DIALOG_DIRECTION_INVALID	Invalid SIP dialog state.
VE_SIP_DIALOG_DIRECTION_INITIATOR	The destination was the initiator of the INVITE that created the SIP dialog.
VE_SIP_DIALOG_DIRECTION_RECIPIENT	The destination was the recipient of the INVITE that created the SIP dialog.
VE_SIP_NUM_DIALOG_DIRECTIONS	The number of elements in this enumeration.

B.203 t_ve_sip_transport_protocol_e

The **t_ve_sip_transport_protocol_e** enumeration is used within the **t_ve_sip_registration_server_s** structure, as described in Appendix B.39.3 on Page 442, and the **t_ve_sip_routing_data_s** structure, as described in Appendix B.42.1 on Page 445. This enumeration specifies the SIP transport type to be used. The **t_ve_sip_transport_protocol_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_TRANSPORT_PROTOCOL_UDP = 0,
  VE_SIP_TRANSPORT_PROTOCOL_TCP,
  VE_SIP_TRANSPORT_PROTOCOL_TCP_TLS
}t_ve_sip_transport_protocol_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_TRANSPORT_PROTOCOL_UDP	The transport type to be used is UDP.
VE_SIP_TRANSPORT_PROTOCOL_TCP	The transport type to be used is TCP.
VE_SIP_TRANSPORT_PROTOCOL_TCP_TLS	The transport type to be used is TCP TLS.

B.204 t_ve_sip_privacy_e

The **t_ve_sip_privacy_e** enumeration is used within the **t_ve_sip_privacy_s** structure, as described in Appendix B.50.1 on Page 465. The enumeration specifies possible SIP privacy settings that allow the application to control the removal of P-Asserted-Identity header fields by the proxy server when transmitting to an element that is not trusted. The **t_ve_sip_privacy_e** enumeration is defined in **ve_sip_defs.h** and is described below:

```
typedef enum{
  VE_SIP_PRIVACY_INVALID=0,
  VE_SIP_PRIVACY_HEADER,
  VE_SIP_PRIVACY_SESSION,
  VE_SIP_PRIVACY_USER,
  VE_SIP_PRIVACY_NONE,
  VE_SIP_PRIVACY_CRITICAL,
  VE_SIP_PRIVACY_ID,
  VE_SIP_NUM_PRIVACY_FIELDS
}t_ve_sip_privacy_e;
```

The enumeration elements are as follows:

Element	Definition
VE_SIP_PRIVACY_INVALID	Not a valid privacy setting.
VE_SIP_PRIVACY_HEADER	Header privacy is required.
VE_SIP_PRIVACY_SESSION	Session privacy is required.
VE_SIP_PRIVACY_USER	User privacy is required.
VE_SIP_PRIVACY_NONE	Privacy not required.
VE_SIP_PRIVACY_CRITICAL	Critical privacy is required (server rejects if it can't provide requested privacy).
VE_SIP_PRIVACY_ID	Identity privacy is required.
VE_SIP_NUM_PRIVACY_FIELDS	The number of elements in this enumeration.

B.205 t_ve_h323_call_forward_reason_e

The **t_ve_h323_call_forward_reason_e** enumeration is returned to calls made to **VE_H323CallForward**, **VE_H323CallForwardActivate** and **VE_H323CallForwardDeactivate**. The enumeration provides reason codes for the forwarding of H.323 calls. The **t_ve_h323_call_forward_reason_e** enumeration is defined in **ve_h323_defs.h** and is described below:

```
typedef enum{
  VE_H323_CALL_FORWARD_TYPE_INVALID=0,
  VE_H323_CALL_FORWARD_TYPE_UNCONDITIONAL,
  VE_H323_CALL_FORWARD_TYPE_BUSY,
  VE_H323_CALL_FORWARD_TYPE_NO_REPLY,
  VE_H323_NUM_CALL_FORWARD_TYPES
}t_ve_h323_call_forward_reason_e;
```

The enumeration elements are as follows:

Element	Definition
VE_H323_CALL_FORWARD_ TYPE_INVALID	The cal forwarding type was invalid.
VE_H323_CALL_FORWARD_ TYPE_UNCONDITIONAL	The call was forwarded because call forwarding is unconditional.
VE_H323_CALL_FORWARD_ TYPE_BUSY	The call was forwarded because the far-end was busy.
VE_H323_CALL_FORWARD_ TYPE_NO_REPLY	The call was forwarded because there was no reply at the far-end.
VE_H323_NUM_CALL_ FORWARD_TYPES	The number of elements in this enumeration.

B.206 t_ve_h323_call_response_e

The **t_ve_h323_call_response_e** enumeration is returned to calls made to **VE_H323CallResponse** and other H.323 response structures. The enumeration provides response codes for the H.323 API function calls. The **t_ve_h323_call_response_e** enumeration is defined in **ve_h323_defs.h** and is described below:

```
typedef enum{
  VE_H323_RESPONSE_INVALID=0,
  VE_H323_RESPONSE_PROCEEDING,
  VE_H323_RESPONSE_ALERTING,
  VE_H323_RESPONSE_CONNECTED,
  VE_H323_RESPONSE_DISCONNECTED_NORMAL,
  VE_H323_RESPONSE_DISCONNECTED_BUSY,
  VE_H323_RESPONSE_DISCONNECTED_REJECT,
  VE_H323_RESPONSE_DISCONNECTED_UNREACHABLE,
  VE_H323_RESPONSE_DISCONNECTED_UNKNOWN,
  VE_H323_RESPONSE_DISCONNECTED_LOCAL,
  VE_H323_RESPONSE_DISCONNECTED_INCOMPLETEADDRESS,
  VE_H323_RESPONSE_TRANSFERING,
  VE_H323_RESPONSE_NO_BANDWIDTH,
  VE_H323_RESPONSE_GATEKEEPER_RESOURCE,
  VE_H323_RESPONSE_UNREACHABLE_DESTINATION,
  VE_H323_RESPONSE_DESTINATION_REJECTION,
  VE_H323_RESPONSE_INVALID_REVISION,
  VE_H323_RESPONSE_NO_PERMISION,
  VE_H323_RESPONSE_UNREACHABLE_GATEKEEPER,
  VE_H323_RESPONSE_GATEWAY_RESOURCE,
  VE_H323_RESPONSE_BAD_FORMAT_ADDRESS,
  VE_H323_RESPONSE_ADAPTIVE_BUSY,
  VE_H323_RESPONSE_IN_CONF,
  VE_H323_RESPONSE_UNDEFINED_REASON,
  VE_H323_RESPONSE_ROUTE_CALL_TO_GATEKEEPER,
  VE_H323_RESPONSE_CALL_FORWARDED,
  VE_H323_RESPONSE_ROUTE_CALL_TO_MC
  VE_H323_RESPONSE_FACILITY_CALL_DEFLECTION,
  VE_H323_RESPONSE_SECURITY_DENIED,
  VE_H323_RESPONSE_CALLED_PARTY_NOT_REGISTERED,
  VE_H323_RESPONSE_CALLER_NOTREGISTERED,
  VE_H323_RESPONSE_NEW_CONNECTION_NEEDED,
  VE_H323_RESPONSE_NO_H245,
  VE_H323_RESPONSE_CONFERENCE_LIST_CHOICE,
  VE_H323_RESPONSE_START_H245,
  VE_H323_NUM_RESPONSES
}t_ve_h323_call_response_e;
```

Continued >>

The enumeration elements of **t_ve_h323_call_response_e** are as follows:

Element	Definition
VE_H323_RESPONSE_ INVALID	The response is invalid.
VE_H323_RESPONSE_ PROCEEDING	The call is proceeding.
VE_H323_RESPONSE_ ALERTING	The call is proceeding, normally this would be when ring-back is played to the user
VE_H323_RESPONSE_ CONNECTED	The call has connected.
VE_H323_RESPONSE_ DISCONNECTED_NORMAL	The call has disconnected normally.
VE_H323_RESPONSE_ DISCONNECTED_BUSY	The call has been disconnected because the called party was busy.
VE_H323_RESPONSE_ DISCONNECTED_REJECT	The call has been disconnected because the called party rejected the call.
VE_H323_RESPONSE_ DISCONNECTED_UNREACHABLE	The call has been disconnected because the specified address was unreachable.
VE_H323_RESPONSE_ DISCONNECTED_UNKNOWN	The call has been disconnected for an unknown reason.
VE_H323_RESPONSE_ DISCONNECTED_LOCAL	The call has been disconnected by the VeriCall Edge user application.
VE_H323_RESPONSE_ DISCONNECTED_INCOMPLETEADDRESS	The call has been disconnected because the address is incomplete.
VE_H323_RESPONSE_ TRANSFERING	The call is being transferred.
VE_H323_RESPONSE_ NO_BANDWIDTH	There is no available bandwidth.
VE_H323_RESPONSE_ GATEKEEPER_RESOURCE	Gatekeeper resources have been exhausted.
VE_H323_RESPONSE_ UNREACHABLE_DESTINATION	The destination cannot be reached.
VE_H323_RESPONSE_ DESTINATION_REJECTION	The destination rejected the transaction request.
VE_H323_RESPONSE_ INVALID_REVISION	Version is not compatible.
VE_H323_RESPONSE_ NO_PERMISION	No permission to perform requested transaction.
VE_H323_RESPONSE_ UNREACHABLE_GATEKEEPER	The destination gatekeeper cannot be reached.
VE_H323_RESPONSE_ GATEWAY_RESOURCE	Gateway resources have been exhausted.
VE_H323_RESPONSE_ BAD_FORMAT_ADDRESS	Destination address is not formatted correctly.
VE_H323_RESPONSE_ ADAPTIVE_BUSY	LAN crowding has caused the call to be dropped.
VE_H323_RESPONSE_ IN_CONF	The destination is busy and cannot respond to the call transaction.
VE_H323_RESPONSE_ UNDEFINED_REASON	Undefined reason for transaction failure.

Continued >>

Element	Definition
VE_H323_RESPONSE_ ROUTE_CALL_TO_GATEKEEPER	Call should be routed to a gatekeeper.
VE_H323_RESPONSE_ CALL_FORWARDED	Call should be forwarded.
VE_H323_RESPONSE_ ROUTE_CALL_TO_MC	Call should be routed to a MC.
VE_H323_RESPONSE_ FACILITY_CALL_DEFLECTION	Call deflection has occurred.
VE_H323_RESPONSE_ SECURITY_DENIED	Access denied.
VE_H323_RESPONSE_ CALLED_PARTY_NOT_REGISTERED	The called party is not registered at the destination.
VE_H323_RESPONSE_ CALLER_NOTREGISTERED	The calling party is not registered.
VE_H323_RESPONSE_ NEW_CONNECTION_NEEDED	The connection failed and a new one should be made.
VE_H323_RESPONSE_ NO_H245	The called party has no H.245 capabilities.
VE_H323_RESPONSE_ CONFERENCE_LIST_CHOICE	Facility message sends conference list choice.
VE_H323_RESPONSE_ START_H245	Request to establish H.245 connection.
VE_H323_NUM_RESPONSES	The number of elements in this enumeration.

B.207 t_ve_h323_reg_response_e

The **t_ve_h323_reg_response_e** enumeration is returned to calls made to the **t_ve_user_msg_h323_registration_response_s** structure as defined in Appendix B.33.36 on Page 394. The enumeration provides response codes for the H.323 API function calls. The **t_ve_h323_reg_response_e** enumeration is defined in **ve_h323_defs.h** and is described below:

```
typedef enum{
  VE_H323_REG_RESPONSE_INVALID=0,
  VE_H323_REG_RESPONSE_REASON_RESOURCE_UNAVAILABLE,
  VE_H323_REG_RESPONSE_REASON_INSUFFICIENT_RESOURCES,
  VE_H323_REG_RESPONSE_REASON_INVALID_REVISION,
  VE_H323_REG_RESPONSE_REASON_INVALID_CALL_SIGNAL_ADDRESS,
  VE_H323_REG_RESPONSE_REASON_INVALID_RAS_ADDRESS,
  VE_H323_REG_RESPONSE_REASON_INVALID_TERMINAL_TYPE,
  VE_H323_REG_RESPONSE_REASON_INVALID_PERMISSION,
  VE_H323_REG_RESPONSE_REASON_INVALID_CONFERENCE_ID,
  VE_H323_REG_RESPONSE_REASON_INVALID_ENDPOINT_ID,
  VE_H323_REG_RESPONSE_REASON_CALLER_NOT_REGISTERED,
  VE_H323_REG_RESPONSE_REASON_CALLED_PARTY_NOT_REGISTERED,
  VE_H323_REG_RESPONSE_REASON_DISCOVERY_REQUIRED,
  VE_H323_REG_RESPONSE_REASON_DUPLICATE_ALIAS,
  VE_H323_REG_RESPONSE_REASON_TRANSPORT_NOT_SUPPORTED,
  VE_H323_REG_RESPONSE_REASON_CALL_IN_PROGRESS,
  VE_H323_REG_RESPONSE_REASON_ROUTE_CALL_TO_GATEKEEPER,
  VE_H323_REG_RESPONSE_REASON_REQUEST_TO_DROP_OTHER,
  VE_H323_REG_RESPONSE_REASON_NOT_REGISTERED,
  VE_H323_REG_RESPONSE_REASON_UNDEFINED,
  VE_H323_REG_RESPONSE_REASON_TERMINAL_EXCLUDED,
  VE_H323_REG_RESPONSE_REASON_NOT_BOUND,
  VE_H323_REG_RESPONSE_REASON_NOT_CURRENTLY_REGISTERED,
  VE_H323_REG_RESPONSE_REASON_REQUEST_DENIED,
  VE_H323_REG_RESPONSE_REASON_LOCATION_NOT_FOUND,
  VE_H323_REG_RESPONSE_REASON_SECURITY_DENIAL,
  VE_H323_REG_RESPONSE_TRANSPORT_QOS_NOT_SUPPORTED,
  VE_H323_REG_RESPONSE_RESOURCE_UNAVAILABLE,
  VE_H323_REG_RESPONSE_INVALID_ALIAS,
  VE_H323_REG_RESPONSE_PERMISSION_DENIED,
  VE_H323_REG_RESPONSE_QOS_CONTROL_NOT_SUPPORTED,
  VE_H323_REG_RESPONSE_INCOMPLETE_ADDRESS,
  VE_H323_REG_RESPONSE_FULL_REGISTRATION_REQUIRED,
  VE_H323_REG_RESPONSE_ROUTE_CALL_TO_SCN,
  VE_H323_REG_RESPONSE_ALIASES_INCONSISTENT,
  VE_H323_REG_RESPONSE_REASON_ADDITIVE_REGISTRATION_NOT_SUPPORTED,
  VE_H323_REG_RESPONSE_REASON_INVALID_TERMINAL_ALIASES,
  VE_H323_REG_RESPONSE_REASON_EXCEEDS_CALL_CAPACITY,
  VE_H323_REG_RESPONSE_REASON_COLLECT_DESTINATION,
  VE_H323_REG_RESPONSE_REASON_COLLECT_PIN,
```

Continued >>

```
    VE_H323_REG_RESPONSE_REASON_GENERIC_DATA,
    VE_H323_REG_RESPONSE_REASON_NEEDED_FEATURE_NOT_SUPPORTED,
    VE_H323_REG_RESPONSE_REASON_UNKNOWN_MESSAGE_RESPONSE,
    VE_H323_REG_RESPONSE_TIME_OUT,
    VE_H323_REG_RESPONSE_OK,
    VE_H323_NUM_REG_RESPONSE
}t_ve_h323_reg_response_e;
```

The enumeration elements of **t_ve_h323_reg_response_e** are as follows:

Element	Definition
VE_H323_REG_RESPONSE_ INVALID	The response was invalid.
VE_H323_REG_RESPONSE_ REASON_RESOURCE_UNAVAILABLE	The resource was unavailable.
VE_H323_REG_RESPONSE_ REASON_INSUFFICIENT_RESOURCES	There were insufficient resources.
VE_H323_REG_RESPONSE_ REASON_INVALID_REVISION	The revision of software is invalid for this operation.
VE_H323_REG_RESPONSE_ REASON_INVALID_CALL_SIGNAL_ADDRESS	The call signal address was invalid.
VE_H323_REG_RESPONSE_ REASON_INVALID_RAS_ADDRESS	The RAS address was invalid.
VE_H323_REG_RESPONSE_ REASON_INVALID_TERMINAL_TYPE	The terminal type was invalid.
VE_H323_REG_RESPONSE_ REASON_INVALID_PERMISSION	Insufficient permissions to perform this task.
VE_H323_REG_RESPONSE_ REASON_INVALID_CONFERENCE_ID	The conference identifier was invalid.
VE_H323_REG_RESPONSE_ REASON_INVALID_ENDPOINT_ID	The data-end point identifier was invalid.
VE_H323_REG_RESPONSE_ REASON_CALLER_NOT_REGISTERED	The caller has not registered with a gatekeeper.
VE_H323_REG_RESPONSE_ REASON_CALLED_PARTY_NOT_REGISTERED	The called party has not registered with a gatekeeper.
VE_H323_REG_RESPONSE_ REASON_DISCOVERY_REQUIRED	Gatekeeper discovery is required before this action can be performed.
VE_H323_REG_RESPONSE_ REASON_DUPLICATE_ALIAS	The alias is a duplicate of another.
VE_H323_REG_RESPONSE_ REASON_TRANSPORT_NOT_SUPPORTED	The defined transport is not supported.
VE_H323_REG_RESPONSE_ REASON_CALL_IN_PROGRESS	There is already a call in progress.
VE_H323_REG_RESPONSE_ REASON_ROUTE_CALL_TO_GATEKEEPER	The call needs to be routed to a gatekeeper.
VE_H323_REG_RESPONSE_ REASON_REQUEST_TO_DROP_OTHER	A request to drop others has been received.
VE_H323_REG_RESPONSE_ REASON_NOT_REGISTERED	The user has not registered with a gatekeeper.

Continued >>

Element	Definition
VE_H323_REG_RESPONSE_ REASON_UNDEFINED	The reason is undefined.
VE_H323_REG_RESPONSE_ REASON_TERMINAL_EXCLUDED	The terminal has been excluded.
VE_H323_REG_RESPONSE_ REASON_NOT_BOUND	The user is not bound.
VE_H323_REG_RESPONSE_ REASON_NOT_CURRENTLY_REGISTERED	The user has not registered with a gatekeeper.
VE_H323_REG_RESPONSE_ REASON_REQUEST_DENIED	The request has been denied.
VE_H323_REG_RESPONSE_ REASON_LOCATION_NOT_FOUND	The location has not been found.
VE_H323_REG_RESPONSE_ REASON_SECURITY_DENIAL	The request has been denied for security reasons.
VE_H323_REG_RESPONSE_ TRANSPORT_QOS_NOT_SUPPORTED	The transport does not support quality of service.
VE_H323_REG_RESPONSE_ RESOURCE_UNAVAILABLE	The resource is unavailable.
VE_H323_REG_RESPONSE_ INVALID_ALIAS	The alias is invalid.
VE_H323_REG_RESPONSE_ PERMISSION_DENIED	Permission has been denied.
VE_H323_REG_RESPONSE_ QOS_CONTROL_NOT_SUPPORTED	The control quality of service is not supported.
VE_H323_REG_RESPONSE_ INCOMPLETE_ADDRESS	The address is incomplete.
VE_H323_REG_RESPONSE_ FULL_REGISTRATION_REQUIRED	Full registration is required.
VE_H323_REG_RESPONSE_ ROUTE_CALL_TO_SCN	The call needs to be routed to SCN.
VE_H323_REG_RESPONSE_ ALIASES_INCONSISTENT	The supplied aliases are inconsistent.
VE_H323_REG_RESPONSE_REASON_ ADDITIVE_REGISTRATION_NOT_SUPPORTED	Additional registrations of this user are not supported.
VE_H323_REG_RESPONSE_ REASON_INVALID_TERMINAL_ALIASES	The terminal aliases are invalid.
VE_H323_REG_RESPONSE_ REASON_EXCEEDS_CALL_CAPACITY	The call exceeds the defined maximum call capacity.
VE_H323_REG_RESPONSE_ REASON_COLLECT_DESTINATION	Unable to collect destination.
VE_H323_REG_RESPONSE_ REASON_COLLECT_PIN	Unable to collect PIN.
VE_H323_REG_RESPONSE_ REASON_GENERIC_DATA	Unable to complete request because of generic data.
VE_H323_REG_RESPONSE_REASON_ NEEDED_FEATURE_NOT_SUPPORTED	The request requires a needed feature that is not supported.
VE_H323_REG_RESPONSE_ REASON_UNKNOWN_MESSAGE_RESPONSE	The response was an unknown message.
VE_H323_REG_RESPONSE_TIME_OUT	The request has timed out.
VE_H323_REG_RESPONSE_OK	The registration request was successful.
VE_H323_NUM_REG_RESPONSE	The number of elements in this enumeration.

B.208 t_ve_h323_call_hold_response_e

The **t_ve_h323_call_hold_response_e** enumeration is returned to calls made to the **t_ve_user_msg_h323_call_hold_response_s** structure as defined in Appendix B.33.46 on Page 401. The enumeration provides response codes for the H.323 API function calls. The **t_ve_h323_call_hold_response_e** enumeration is defined in **ve_h323_defs.h** and is described below:

```
typedef enum{
  VE_H323_CALL_HOLD_RESPONSE_INVALID=0,
  VE_H323_CALL_HOLD_RESPONSE_OK,
  VE_H323_CALL_HOLD_RESPONSE_FAILED,
  VE_H323_NUM_CALL_HOLD_RESPONSES
}t_ve_h323_call_hold_response_e;
```

The enumeration elements of **t_ve_h323_call_hold_response_e** are as follows:

Element	Definition
VE_H323_CALL_HOLD_RESPONSE_INVALID	The call hold response is invalid.
VE_H323_CALL_HOLD_RESPONSE_OK	The call hold request is proceeding.
VE_H323_CALL_HOLD_RESPONSE_FAILED	The call hold request failed.
VE_H323_NUM_CALL_HOLD_RESPONSES	The number of elements in this enumeration.

B.209 t_ve_h323_mwi_response_e

The **t_ve_h323_mwi_response_e** enumeration is returned to calls made to the **t_ve_user_msg_h323_mwi_response_s** structure as defined in Appendix B.33.47 on Page 401. The enumeration provides response codes for the H.323 API function calls. The **t_ve_h323_mwi_response_e** enumeration is defined in **ve_h323_defs.h** and is described below:

```
typedef enum{
  VE_H323_MWI_RESPONSE_INVALID=0,
  VE_H323_MWI_RESPONSE_OK,
  VE_H323_MWI_RESPONSE_FAILED,
  VE_H323_NUM_MWI_RESPONSES
}t_ve_h323_mwi_response_e;
```

The enumeration elements of **t_ve_h323_mwi_response_e** are as follows:

Element	Definition
VE_H323_MWI_RESPONSE_INVALID	The message waiting request response is invalid.
VE_H323_MWI_RESPONSE_OK	The message waiting request is proceeding.
VE_H323_MWI_RESPONSE_FAILED	The message waiting request failed.
VE_H323_NUM_MWI_RESPONSES	The number of elements in this enumeration.

B.210 t_ve_h323_transfer_response_e

The **t_ve_h323_transfer_response_e** enumeration is returned to calls made to the **t_ve_user_msg_h323_transfer_response_s** structure as defined in Appendix B.33.42 on Page 398. The enumeration provides response codes for the H.323 API function calls. The **t_ve_h323_transfer_response_e** enumeration is defined in **ve_h323_defs.h** and is described below:

```
typedef enum{
  VE_H323_TRANSFER_RESPONSE_INVALID=0,
  VE_H323_TRANSFER_RESPONSE_IDENTIFY,
  VE_H323_TRANSFER_RESPONSE_ABANDON,
  VE_H323_TRANSFER_RESPONSE_INITIATE,
  VE_H323_TRANSFER_RESPONSE_SETUP,
  VE_H323_TRANSFER_RESPONSE_ACTIVE,
  VE_H323_TRANSFER_RESPONSE_COMPLETE,
  VE_H323_TRANSFER_RESPONSE_UPDATE,
  VE_H323_TRANSFER_RESPONSE_SUBADDRESSTRANSFER,
  VE_H323_TRANSFER_RESPONSE_INVALIDREROUTINGNUMBER,
  VE_H323_TRANSFER_RESPONSE_UNRECOGNIZEDCALLIDENTITY,
  VE_H323_TRANSFER_RESPONSE_ESTABLISHMENTFAILURE,
  VE_H323_TRANSFER_RESPONSE_UNSPECIFIED,
  VE_H323_TRANSFER_RESPONSE_TIMEOUT,
  VE_H323_TRANSFER_RESPONSE_IDENTIFY_FAILED,
  VE_H323_TRANSFER_RESPONSE_INITIATE_FAILED,
  VE_H323_TRANSFER_RESPONSE_SETUP_FAILED,
  VE_H323_TRANSFER_RESPONSE_COMPLETE_FAILED,
  VE_H323_NUM_TRANSFER_RESPONSES
}t_ve_h323_transfer_response_e;
```

The enumeration elements are as follows:

Element	Definition
VE_H323_TRANSFER_RESPONSE_ INVALID	The call transfer response was invalid.
VE_H323_TRANSFER_RESPONSE_ IDENTIFY	The call transfer is in the Identify state.
VE_H323_TRANSFER_RESPONSE_ ABANDON	The call transfer has been abandoned.
VE_H323_TRANSFER_RESPONSE_ INITIATE	The call transfer is in the Initiate state.
VE_H323_TRANSFER_RESPONSE_ SETUP	The call transfer is in the Setup state.
VE_H323_TRANSFER_RESPONSE_ ACTIVE	The call transfer is in the Active state.
VE_H323_TRANSFER_RESPONSE_ COMPLETE	The call transfer has been completed.
VE_H323_TRANSFER_RESPONSE_ UPDATE	The call transfer has been updated.
VE_H323_TRANSFER_RESPONSE_ SUBADDRESSTRANSFER	The call transfer is in the Sub Address transfer state.

Continued >>

Element	Definition
VE_H323_TRANSFER_RESPONSE_ INVALIDREROUTINGNUMBER	The call transfer request has been abandoned due to an invalid number.
VE_H323_TRANSFER_RESPONSE_ UNRECOGNIZEDCALLIDENTITY	The call transfer request has been abandoned due to an unrecognized call identity.
VE_H323_TRANSFER_RESPONSE_ ESTABLISHMENTFAILURE	The call transfer has been abandoned due to an establishment failure.
VE_H323_TRANSFER_RESPONSE_ UNSPECIFIED	The call transfer response is unspecified.
VE_H323_TRANSFER_RESPONSE_ TIMEOUT	The call transfer has timed out.
VE_H323_TRANSFER_RESPONSE_ IDENTIFY_FAILED	The call transfer has failed to enter the identify state.
VE_H323_TRANSFER_RESPONSE_ INITIATE_FAILED	The call transfer initiation has failed.
VE_H323_TRANSFER_RESPONSE_ SETUP_FAILED	The call transfer has failed to be set-up.
VE_H323_TRANSFER_RESPONSE_ COMPLETE_FAILED	The call transfer has failed to complete successfully.
VE_H323_NUM_TRANSFER_RESPONSES	The number of elements in this enumeration.

B.211 t_ve_h323_forward_response_e

The **t_ve_h323_forward_response_e** enumeration is returned to calls made to the **t_ve_user_msg_h323_forward_response_s** structure as defined in Appendix B.33.44 on Page 399. The enumeration provides response codes for the H.323 API function calls. The **t_ve_h323_forward_response_e** enumeration is defined in **ve_h323_defs.h** and is described below:

```
typedef enum{
  VE_H323_FORWARD_RESPONSE_INVALID=0,
  VE_H323_FORWARD_RESPONSE_OK,
  VE_H323_FORWARD_RESPONSE_FAIL,
  VE_H323_NUM_FORWARD_RESPONSES
}t_ve_h323_forward_response_e;
```

The enumeration elements of **t_ve_h323_forward_response_e** are as follows:

Element	Definition
VE_H323_FORWARD_RESPONSE_INVALID	The call forward response is invalid.
VE_H323_FORWARD_RESPONSE_OK	The call forward request is proceeding.
VE_H323_FORWARD_RESPONSE_FAILED	The call forward request failed.
VE_H323_FORWARD_HOLD_RESPONSES	The number of elements in this enumeration.

B.212 t_ve_h323_request_type_e

The **t_ve_h323_request_type_e** enumeration is part of the **t_ve_user_msg_h323_request_failed_s** structure, as defined in Appendix B.33.49 on Page 403. The enumeration provides response identification of the H.323 request type. The **t_ve_h323_request_type_e** enumeration is defined in **ve_h323_defs.h** and is described below:

```
typedef enum{
  VE_H323_REQUEST_TYPE_INVALID=0,
  VE_H323_REQUEST_TYPE_VE_H323STARTCALL,
  VE_H323_REQUEST_TYPE_VE_H323STOPCALL,
  VE_H323_REQUEST_TYPE_VE_H323CALLRESPONSE,
  VE_H323_REQUEST_TYPE_VE_H323UPDATECALL,
  VE_H323_REQUEST_TYPE_VE_H323ADDUSERSREGISTRATIONDATA,
  VE_H323_REQUEST_TYPE_VE_H323UPDATEUSERSREGISTRATIONDATA,
  VE_H323_REQUEST_TYPE_VE_H323REMOVEUSERSREGISTRATIONDATA,
  VE_H323_REQUEST_TYPE_VE_H323GETUSERSREGISTRATIONDATA,
  VE_H323_REQUEST_TYPE_VE_H323REGISTERUSER,
  VE_H323_REQUEST_TYPE_VE_H323DEREGISTERUSER,
  VE_H323_REQUEST_TYPE_VE_H323SENDUSERINPUT,
  VE_H323_REQUEST_TYPE_VE_H323VIDEOFASTUPDATE,
  VE_H323_REQUEST_TYPE_VE_H323VIDEOFREEZE,
  VE_H323_REQUEST_TYPE_VE_H323CALLTRANSFERCONSULT,
  VE_H323_REQUEST_TYPE_VE_H323CALLTRANSFERDONTCONSULT,
  VE_H323_REQUEST_TYPE_VE_H323CALLFORWARD,
  VE_H323_REQUEST_TYPE_VE_H323CALLFORWARDACTIVATE,
  VE_H323_REQUEST_TYPE_VE_H323CALLFORWARDDEACTIVATE,
  VE_H323_REQUEST_TYPE_VE_H323CALLHOLDON,
  VE_H323_REQUEST_TYPE_VE_H323CALLHOLDOFF,
  VE_H323_REQUEST_TYPE_VE_H323MWIINTERROGATE,
  VE_H323_NUM_REQUEST_TYPES
}t_ve_h323_request_type_e;
```

Continued >>

The enumeration elements of **t_ve_h323_request_type_e** are as follows:

Element	Definition
VE_H323_REQUEST_TYPE_INVALID	The request type was invalid.
VE_H323_REQUEST_TYPE_VE_H323STARTCALL	The request type was **VE_H323StartCall**.
VE_H323_REQUEST_TYPE_VE_H323STOPCALL	The request type was **VE_H323StopCall**.
VE_H323_REQUEST_TYPE_VE_H323CALLRESPONSE	The request type was **VE_H323CallResponse**.
VE_H323_REQUEST_TYPE_VE_H323UPDATECALL	The request type was **VE_H323UpdateCall**.
VE_H323_REQUEST_TYPE_VE_H323ADDUSERSREGISTRATIONDATA	The request type was **VE_H323AddUsersRegistrationData**.
VE_H323_REQUEST_TYPE_VE_H323UPDATEUSERSREGISTRATIONDATA	The request type was **VE_H323UpdateUsersRegistrationData**.
VE_H323_REQUEST_TYPE_VE_H323REMOVEUSERSREGISTRATIONDATA	The request type was **VE_H323RemoveUsersRegistrationData**.
VE_H323_REQUEST_TYPE_VE_H323GETUSERSREGISTRATIONDATA	The request type was **VE_H323GetUsersRegistrationData**.
VE_H323_REQUEST_TYPE_VE_H323REGISTERUSER	The request type was **VE_SIPRegisterUser**.
VE_H323_REQUEST_TYPE_VE_H323DEREGISTERUSER	The request type was **VE_H323DeregisterUser**.
VE_H323_REQUEST_TYPE_VE_H323SENDUSERINPUT	The request type was **VE_H323SendUserInput**.
VE_H323_REQUEST_TYPE_VE_H323VIDEOFASTUPDATE	The request type was **VE_H323VideoFastUpdate**
VE_H323_REQUEST_TYPE_VE_H323VIDEOFREEZE	The request type was **VE_H323VideoFreeze**
VE_H323_REQUEST_TYPE_VE_H323CALLTRANSFERCONSULT	The request type was **VE_H323CallTransferConsult**.
VE_H323_REQUEST_TYPE_VE_H323CALLTRANSFERDONTCONSULT	The request type was **VE_H323CallTransferDontConsult**.
VE_H323_REQUEST_TYPE_VE_H323CALLFORWARD	The request type was **VE_H323CallForward**.
VE_H323_REQUEST_TYPE_VE_H323CALLFORWARDACTIVATE	The request type was **VE_H323CallForwardActivate**.
VE_H323_REQUEST_TYPE_VE_H323CALLFORWARDDEACTIVATE	The request type was **VE_H323CallForwardDeactivate**.
VE_H323_REQUEST_TYPE_VE_H323CALLHOLDON	The request type was **VE_H323CallHoldOn**.
VE_H323_REQUEST_TYPE_VE_H323CALLHOLDOFF	The request type was **VE_H323CallHoldOff**.
VE_H323_REQUEST_TYPE_VE_H323MWIINTERROGATE	The request type was **VE_H323MWIInterrogate**.
VE_H323_NUM_REQUEST_TYPES	The number of elements in this enumeration.

B.213 t_ve_h323_request_failed_reason_e

The **t_ve_h323_request_failed_reason_e** enumeration is part of the t_ve_user_msg_h323_request_failed_s structure, as defined in Appendix B.33.49 on Page 403. The enumeration provides reason codes for failed H.323 API function calls. The **t_ve_h323_request_failed_reason_e** enumeration is defined in **ve_h323_defs.h** and is described below:

```
typedef enum{
  VE_H323_REQUEST_FAILED_REASON_INVALID=0,
  VE_H323_REQUEST_FAILED_INVALID_USER_HANDLE,
  VE_H323_REQUEST_FAILED_INVALID_H323_CALL_HANDLE,
  VE_H323_REQUEST_FAILED_INVALID_RESPONSE,
  VE_H323_REQUEST_FAILED_CAPABILITY_EXCHANGE,
  VE_H323_REQUEST_FAILED_INSUFFICIENT_BANDWIDTH,
  VE_H323_REQUEST_FAILED_NO_FREE_CALL_OBJECTS,
  VE_H323_REQUEST_FAILED_TRANSPORT_ERROR,
  VE_H323_REQUEST_FAILED_FUNCTION_NOT_VALID,
  VE_H323_REQUEST_FAILED_INVALID_H323_ADDRESS,
  VE_H323_REQUEST_FAILED_INTERNAL_ERROR,
  VE_H323_NUM_REQUEST_FAILED_REASONS
}t_ve_h323_request_failed_reason_e;
```

The enumeration elements are as follows:

Element	Definition
VE_H323_REQUEST_FAILED_ REASON_INVALID	The reason the request failed is invalid.
VE_H323_REQUEST_FAILED_ INVALID_USER_HANDLE	The reason the request failed was that the user handle was invalid.
VE_H323_REQUEST_FAILED_ INVALID_H323_CALL_HANDLE	The reason the request failed was that the H.323 call handle was invalid.
VE_H323_REQUEST_FAILED_ INVALID_RESPONSE	The reason the request failed was that the response was invalid.
VE_H323_REQUEST_FAILED_ CAPABILITY_EXCHANGE	The reason the request failed was that there were no matching capabilities available.
VE_H323_REQUEST_FAILED_ INSUFFICIENT_BANDWIDTH	The H.323 stack was unable to complete the call because there is insufficient bandwidth available.
VE_H323_REQUEST_FAILED_ NO_FREE_CALL_OBJECTS	The reason the request failed was that there were no free H.323 call objects available.
VE_H323_REQUEST_FAILED_ TRANSPORT_ERROR	The reason the request failed was that there was a H.323 transport error.
VE_H323_REQUEST_FAILED_ FUNCTION_NOT_VALID	The reason the request failed was that the H.323 function call was not valid. An example would be trying to stop a call before it is active.
VE_H323_REQUEST_FAILED_ INVALID_H323_ADDRESS	The reason the request failed was that the H.323 address was invalid.
VE_H323_REQUEST_FAILED_ INTERNAL_ERROR	The reason the request failed was that there was a H.323 stack internal error.
VE_H323_NUM_REQUEST_ FAILED_REASONS	The number of elements in this enumeration.

B.214 t_ve_h323_name_id_char_set_e

The **t_ve_h323_name_id_char_set_e** enumeration is part of the **t_ve_h323_name_id_s** structure, as defined in Appendix B.57 on Page 474. The enumeration provides H.323 name character-set types. The **t_ve_h323_name_id_char_set_e** enumeration is defined in **ve_h323_defs.h** and is described below:

```
typedef enum{
  VE_H323_NAME_ID_CHAR_SET_UNKNOWN=0,
  VE_H323_NAME_ID_CHAR_SET_ISO8859_1,
  VE_H323_NAME_ID_CHAR_SET_T_61,
  VE_H323_NAME_ID_CHAR_SET_ISO8859_2,
  VE_H323_NAME_ID_CHAR_SET_ISO8859_3,
  VE_H323_NAME_ID_CHAR_SET_ISO8859_4,
  VE_H323_NAME_ID_CHAR_SET_ISO8859_5,
  VE_H323_NAME_ID_CHAR_SET_ISO8859_7,
  VE_H323_NUM_NAME_ID_CHAR_SETS
}t_ve_h323_name_id_char_set_e;
```

The enumeration elements are as follows:

Element	Definition
VE_H323_NAME_ID_ CHAR_SET_UNKNOWN	The character set is unknown.
VE_H323_NAME_ID_ CHAR_SET_ISO8859_1	The character set is ISO_08859.1.
VE_H323_NAME_ID_ CHAR_SET_T_61	The character set is T_61.
VE_H323_NAME_ID_ CHAR_SET_ISO8859_2	The character set is ISO_08859.2.
VE_H323_NAME_ID_ CHAR_SET_ISO8859_3	The character set is ISO_08859.3.
VE_H323_NAME_ID_ CHAR_SET_ISO8859_4	The character set is ISO_08859.4.
VE_H323_NAME_ID_ CHAR_SET_ISO8859_5	The character set is ISO_08859.5.
VE_H323_NAME_ID_ CHAR_SET_ISO8859_7	The character set is ISO_08859.7.
VE_H323_NUM_NAME_ID_ CHAR_SETS	The number of elements in this enumeration.

B.215 t_ve_h323_name_id_restriction_e

The **t_ve_h323_name_id_restriction_e** enumeration is part of the **t_ve_h323_name_id_s** structure, as defined in Appendix B.57 on Page 474. The enumeration provides H.323 name restriction types. The **t_ve_h323_name_id_restriction_e** enumeration is defined in **ve_h323_defs.h** and is described below:

```
typedef enum{
  VE_H323_NAME_ID_RESTRICTION_NAME_NOT_AVAILABLE=0,
  VE_H323_NAME_ID_RESTRICTION_NAME_PRESENTATION_ALLOWED,
  VE_H323_NAME_ID_RESTRICTION_NAME_PRESENTATION_RESTRICTED,
  VE_H323_NUM_NAME_ID_RESTRICTION
}t_ve_h323_name_id_restriction_e;
```

The enumeration elements are as follows:

Element	Definition
VE_H323_NAME_ID_RESTRICTION_ NAME_NOT_AVAILABLE	Restrictions to the H.323 user name are not available.
VE_H323_NAME_ID_RESTRICTION_ NAME_PRESENTATION_ALLOWED	There are no restrictions to the H.323 user name.
VE_H323_NAME_ID_RESTRICTION_ NAME_PRESENTATION_RESTRICTED	Restrictions to the H.323 user name are applicable.
VE_H323_NUM_NAME_ID_RESTRICTIONS	The number of elements in this enumeration.

B.216 t_ve_h323_call_waiting_flag_e

The **t_ve_h323_call_waiting_flag_e** enumeration is returned to calls made to **VE_H323CallResponse** and the structure **t_ve_user_msg_h323_call_response_s**, as defined in Appendix B.33.41 on Page 397. The enumeration provides H.323 with the ability to indicate call waiting. The **t_ve_h323_call_waiting_flag_e** enumeration is defined in **ve_h323_defs.h** and is described below:

```
typedef enum{
  VE_H323_CALL_WAITING_FLAG_INVALID=0,
  VE_H323_CALL_WAITING_FLAG_TRUE,
  VE_H323_CALL_WAITING_FLAG_FALSE,
  VE_H323_NUM_CALL_WAITING_FLAG
}t_ve_h323_call_waiting_flag_e;
```

The enumeration elements are as follows:

Element	Definition
VE_H323_CALL_WAITING_ FLAG_INVALID	The H.323 call waiting flag is invalid.
VE_H323_CALL_WAITING_ FLAG_TRUE	The H.323 call waiting flag is enabled.
VE_H323_CALL_WAITING_ FLAG_FALSE	The H.323 call waiting flag is disabled.
VE_H323_NUM_CALL_WAITING_FLAG	The number of elements in this enumeration.

B.217 t_ve_h323_q931_annex_e_protocol_preference_e

The **t_ve_h323_q931_annex_e_protocol_preference_e** enumeration is returned to calls made to **t_ve_h323_q931_annex_e_configuration_s** the structure as defined in Appendix B.58.3.1 on Page 481. The enumeration provides H.323 Q.931 protocol to be defined. The **t_ve_h323_q931_annex_e_protocol_preference_e** enumeration is defined in **ve_h323_configuration.h** and is described below:

```
typedef enum{
  VE_H323_Q931_ANNEX_E_PROTOCOL_PREFERENCE_INVALID=0,
  VE_H323_Q931_ANNEX_E_PROTOCOL_PREFERENCE_ANNEXE,
  VE_H323_Q931_ANNEX_E_PROTOCOL_PREFERENCE_TPKT,
  VE_H323_Q931_ANNEX_E_PROTOCOL_PREFERENCE_RACEANNEXE,
  VE_H323_Q931_ANNEX_E_PROTOCOL_PREFERENCE_RACETPKT,
  VE_NUM_H323_Q931_ANNEX_E_PROTOCOL_PREFERENCES
}t_ve_h323_q931_annex_e_protocol_preference_e;
```

The enumeration elements are as follows:

Element	Definition
VE_H323_Q931_ANNEX_E_PROTOCOL_ PREFERENCE_INVALID	The Q.931 Annex E protocol preference was invalid.
VE_H323_Q931_ANNEX_E_PROTOCOL_ PREFERENCE_ANNEXE	The Q.931 Annex E protocol preference is Annex-E.
VE_H323_Q931_ANNEX_E_PROTOCOL_ PREFERENCE_TPKT	The Q.931 Annex E protocol preference is TPKT.
VE_H323_Q931_ANNEX_E_PROTOCOL_ PREFERENCE_RACEANNEXE	The Q.931 Annex E protocol preference is Race-Annex-E.
VE_H323_Q931_ANNEX_E_PROTOCOL_ PREFERENCE_RACETPKT	The Q.931 Annex E protocol preference is Race-TPKT.
VE_NUM_H323_Q931_ANNEX_E_ PROTOCOL_PREFERENCES	The number of elements in this enumeration.

B.218 t_ve_h323_gatekeeper_discovery_e

The **t_ve_h323_gatekeeper_discovery_e** enumeration is part of the **t_ve_h323_ras_configuration_s** structure, as defined in Appendix B.58.2 on Page 477. The enumeration provides H.323 the method of discovering available gatekeepers. The **t_ve_h323_gatekeeper_discovery_e** enumeration is defined in **ve_h323_configuration.h** and is described below:

```
typedef enum{
  VE_H323_GATEKEEPER_DISCOVERY_INVALID=0,
  VE_H323_GATEKEEPER_DISCOVERY_NONE,
  VE_H323_GATEKEEPER_DISCOVERY_AUTO,
  VE_H323_GATEKEEPER_DISCOVERY_MANUAL,
  VE_H323_NUM_GATEKEEPER_DISCOVERY
}t_ve_h323_gatekeeper_discovery_e;
```

The enumeration elements are as follows:

Element	Definition
VE_H323_GATEKEEPER_DISCOVERY_INVALID	The gatekeeper discovery is invalid.
VE_H323_GATEKEEPER_DISCOVERY_NONE	Disable RAS.
VE_H323_GATEKEEPER_DISCOVERY_AUTO	Gatekeeper discovery is performed automatically.
VE_H323_GATEKEEPER_DISCOVERY_MANUAL	Gatekeeper discovery is disabled and the gatekeeper address will be supplied by the user.
VE_H323_NUM_GATEKEEPER_DISCOVERY	The number of elements in this enumeration.

B.219 t_ve_h323_video_feature_request_e

The **t_ve_h323_video_feature_request_e** enumeration is returned to calls made to **t_ve_user_msg_h323_video_feature_request_s** as defined in Appendix B.33.50 on Page 404. The enumeration provides a response to a requested H.323 video command request. The **t_ve_h323_video_feature_request_e** enumeration is defined in **ve_h323_defs.h** and is described below:

```
typedef enum{
  VE_H323_VIDEO_FEATURE_REQUEST_INVALID=0,
  VE_H323_VIDEO_FEATURE_REQUEST_FREEZE_PICTURE,
  VE_H323_NUM_VIDEO_FEATURE_REQUEST
}t_ve_h323_video_feature_request_e;
```

The enumeration elements are as follows:

Element	Definition
VE_H323_VIDEO_FEATURE_REQUEST_INVALID	Message indicating the request was invalid.
VE_H323_VIDEO_FEATURE_REQUEST_FREEZE_PICTURE	Message indicating the request for a H.323 video freeze.
VE_H323_NUM_VIDEO_FEATURE_REQUEST	The number of elements in this enumeration.

B.220 t_ve_h323_call_start_method_e

The **t_ve_h323_call_start_method_e** enumeration used within the **t_ve_h323_system_configuration_s** structure as defined in Appendix B.58.1 on Page 476. The enumeration defines the H.323 call start method to be used, allowing the fast call start method to be enabled/disabled. The **t_ve_h323_call_start_method_e** enumeration is defined in **ve_h323_configuration.h** and is described below:

```
typedef enum{
  VE_H323_CALL_START_METHOD_INVALID=0,
  VE_H323_CALL_START_METHOD_NORMAL_START,
  VE_H323_CALL_START_METHOD_FAST_START,
  VE_H323_NUM_CALL_START_METHODS
}t_ve_h323_call_start_method_e;
```

The enumeration elements are as follows:

Element	Definition
VE_H323_CALL_START_METHOD_INVALID	This is an invalid start method selection.
VE_H323_CALL_START_METHOD_NORMAL_START	H.323 call start is a normal start.
VE_H323_CALL_START_METHOD_FAST_START	H.323 call start is a fast start.
VE_H323_NUM_CALL_START_METHODS	The number of elements in this enumeration.

B.221 t_ve_h323_hold_method_e

The **t_ve_h323_hold_method_e** enumeration is used within the **VE_H323CallHoldOn** function. The enumeration provides the methods available to the function to place the outgoing call on hold. The **t_ve_h323_hold_method_e** enumeration is defined in **ve_h323_defs.h** and is described below:

```
typedef enum{
  VE_H323_HOLD_METHOD_INVALID=0,
  VE_H323_HOLD_METHOD_H450_4,
  VE_H323_HOLD_METHOD_EMPTY_CAPS,
  VE_H323_NUM_HOLD_METHODS
}t_ve_h323_hold_method_e;
```

The enumeration elements are as follows:

Element	Definition
VE_H323_HOLD_METHOD_INVALID	The call hold method is invalid.
VE_H323_HOLD_METHOD_H450_4	The call hold method to be used is the H.450.4 method.
VE_H323_HOLD_METHOD_EMPTY_CAPS	The call hold method to be used is the empty capabilities method.
VE_H323_NUM_HOLD_METHODS	The number of elements in this enumeration.

B.222 t_ve_h323_user_sub_channel_select_e

The **t_ve_h323_user_sub_channel_select_e** enumeration is used within the **t_ve_h323_user_channel_selection_s** structure, described in Appendix B.56.2 on Page 472. The enumeration provides the H.323 sub-channel type. The **t_ve_h323_user_sub_channel_select_e** enumeration is defined in **ve_h323_defs.h** and is described below:

```
typedef enum{
  VE_H323_SUB_CHANNEL_SELECT_ILLEGAL,
  VE_H323_SUB_CHANNEL_SELECT_PRIMARY,
  VE_H323_SUB_CHANNEL_SELECT_SECONDARY,
  VE_H323_SUB_CHANNEL_SELECT_COUNT
}t_ve_h323_user_sub_channel_select_e;
```

The enumeration elements are as follows:

Element	Definition
VE_H323_SUB_CHANNEL_SELECT_ILLEGAL	This element is used internally by the VeriCall Edge system and should not be used by the user application.
VE_H323_SUB_CHANNEL_SELECT_PRIMARY	The sub-channel is a primary channel.
VE_H323_SUB_CHANNEL_SELECT_SECONDARY	The sub channel is a secondary channel.
VE_H323_SUB_CHANNEL_SELECT_COUNT	The number of elements in this enumeration.

B.223 t_ve_user_msg_play_prompt_event_e

The **t_ve_user_msg_play_prompt_event_e** enumeration is part of the **t_ve_user_msg_play_prompt_event_s** structure as defined in Appendix B.33.56 on Page 407. The enumeration provides a response to a requested file played into the audio stream. The **t_ve_user_msg_play_prompt_event_e** enumeration is defined in **ve_events.h** and is described below:

```
typedef enum{
  VE_USER_MSG_PLAY_PROMPT_COMPLETE = 0,
  VE_USER_MSG_PLAY_PROMPT_CANNOT_OPEN_FILE = 1,
}t_ve_user_msg_play_prompt_event_e;
```

The enumeration elements are as follows:

Element	Definition
VE_USER_MSG_PLAY_PROMPT_COMPLETE	The file was successfully played to the audio stream.
VE_USER_MSG_PLAY_PROMPT_CANNOT_OPEN_FILE	The specified file cannot be found.

B.224 t_ve_user_msg_record_prompt_event_e

The **t_ve_user_msg_record_prompt_event_e** enumeration is part of the **t_ve_user_msg_record_prompt_event_s** structure as defined in Appendix B.33.57 on Page 407. The enumeration provides a response to a requested file recorded from an audio stream. The **t_ve_user_msg_record_prompt_event_e** enumeration is defined in **ve_events.h** and is described below:

```
typedef enum{
  VE_USER_MSG_RECORD_PROMPT_COMPLETE = 0,
  VE_USER_MSG_RECORD_PROMPT_CANNOT_OPEN_FILE = 1,
}t_ve_user_msg_record_prompt_event_e;
```

The enumeration elements are as follows:

Element	Definition
VE_USER_MSG_RECORD_PROMPT_COMPLETE	The file was successfully recorded from the audio stream.
VE_USER_MSG_RECORD_PROMPT_CANNOT_OPEN_FILE	The specified file cannot be found.

B.225 t_ve_user_msg_encode_image_event_e

The **t_ve_user_msg_encode_image_event_e** enumeration is part of the **t_ve_user_msg_encode_image_event_s** structure as defined in Appendix B.33.58 on Page 408. The enumeration provides a response to a requested video image encode command. The **t_ve_user_msg_encode_image_event_e** enumeration is defined in **ve_events.h** and is described below:

```
typedef enum{
  VE_USER_MSG_ENCODE_IMAGE_COMPLETE = 0,
  VE_USER_MSG_ENCODE_IMAGE_CANNOT_OPEN_FILE,
  VE_USER_MSG_ENCODE_IMAGE_ENCODE_ERROR,
  VE_USER_MSG_ENCODE_IMAGE_ENCODE_IN_PROGRESS
}t_ve_user_msg_encode_image_event_e;
```

The enumeration elements are as follows:

Element	Definition
VE_USER_MSG_ENCODE_IMAGE_COMPLETE	The file was successfully recorded from the audio stream.
VE_USER_MSG_ENCODE_IMAGE_CANNOT_OPEN_FILE	The file to which the video image is to be written to cannot be opened.
VE_USER_MSG_ENCODE_IMAGE_ENCODE_ERROR	An unspecified error occurred while attempting to encode the video image.
VE_USER_MSG_ENCODE_IMAGE_ENCODE_IN_PROGRESS	The video image is still in the process of being encoded.

B.226 t_ve_encode_image_event_type_e

The **t_ve_encode_image_event_type_e** enumeration is part of the **t_ve_user_msg_encode_image_event_s** structure as defined in Appendix B.33.58 on Page 408. The enumeration provides a response to determine whether the result if for a local or received image. The **t_ve_encode_image_event_type_e** enumeration is defined in **ve_events.h** and is described below:

```
typedef enum{
  VE_ENCODE_IMAGE_EVENT_TYPE_ILLEGAL = 0,
  VE_ENCODE_IMAGE_EVENT_TYPE_LOCAL_IMAGE,
  VE_ENCODE_IMAGE_EVENT_TYPE_RX_IMAGE
}t_ve_encode_image_event_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_ENCODE_IMAGE_EVENT_TYPE_ILLEGAL	The event is of an illegal type.
VE_ENCODE_IMAGE_EVENT_TYPE_LOCAL_IMAGE	The event refers to a local image.
VE_ENCODE_IMAGE_EVENT_TYPE_RX_IMAGE	The event refers to a received image.

B.227 t_ve_nat_discover_method_e

The **t_ve_nat_discover_method_e** enumeration is part of the **t_ve_nat_init_s** structure as described in Appendix B.62 on Page 494. The enumeration provides the NAT traversal module discovery method to be used. The **t_ve_nat_discover_method_e** enumeration is defined in **ve_nat_defs.h** and is described below:

```
typedef enum{
  VE_NAT_DISCOVER_METHOD_INVALID=0,
  VE_NAT_DISCOVER_METHOD_RFC3489,
  VE_NUM_NAT_DISCOVER_METHODS
}t_ve_nat_discover_method_e;
```

The enumeration elements are as follows:

Element	Definition
VE_NAT_DISCOVER_METHOD_INVALID	Used internally by the VeriCall Edge system and should not be used by the user application.
VE_NAT_DISCOVER_METHOD_RFC3489	Use the RFC3489 (STUN) discovery method.
VE_NUM_NAT_DISCOVER_METHODS	Used internally by the VeriCall Edge system and should not be used by the user application.

B.228 t_ve_nat_state_e

The **t_ve_nat_state_e** enumeration is used within the following structures:

- **t_ve_nat_media_addresses_s** described in Appendix B.61.1 on Page 493.
- **t_ve_nat_stun_init_s** described in Appendix B.62.1 on Page 495.
- **t_ve_nat_turn_init_s** described in Appendix B.62.2 on Page 495.
- **t_ve_nat_configuration_s** described in Appendix B.63 on Page 496.
- **t_ve_nat_sip_addresses_s** described in Appendix B.63.1 on Page 497.

The enumeration allows specific NAT traversal module configuration or functional parameters to be enabled/disabled. The **t_ve_nat_state_e** enumeration is defined in **ve_nat_defs.h** and is described below:

```
typedef enum{
  VE_NAT_STATE_INVALID=0,
  VE_NAT_STATE_ENABLED,
  VE_NAT_STATE_DISABLED,
  VE_NUM_NAT_STATES
}t_ve_nat_state_e;
```

The enumeration elements are as follows:

Element	Definition
VE_NAT_STATE_INVALID	Used internally by the VeriCall Edge system and should not be used by the user application.
VE_NAT_STATE_ENABLED	The NAT traversal modules is currently enabled.
VE_NAT_STATE_DISABLED	The NAT traversal modules is currently disabled.
VE_NUM_NAT_STATES	Used internally by the VeriCall Edge system and should not be used by the user application.

B.229 t_ve_nat_types_e

The **t_ve_nat_types_e** enumeration is part of the **t_ve_nat_configuration_s** described in Appendix B.63 on Page 496. The enumeration provides the type of NAT device behind which the VeriCall Edge system is sitting. The **t_ve_nat_types_e** enumeration is defined in **ve_nat_defs.h** and is described below:

```
typedef enum{
  VE_NAT_TYPE_INVALID=0,
  VE_NAT_TYPE_OPEN,
  VE_NAT_TYPE_FULL_CONE,
  VE_NAT_TYPE_ADDRESS_RESTRICTED_CONE,
  VE_NAT_TYPE_PORT_RESTRICTED_CONE,
  VE_NAT_TYPE_SYMMETRIC_NAT,
  VE_NAT_TYPE_SYMMETRIC_FIREWALL,
  VE_NAT_TYPE_UDP_BLOCKED,
  VE_NUM_NAT_TYPES
}t_ve_nat_types_e;
```

The enumeration elements are as follows:

Element	Definition
VE_NAT_TYPE_INVALID	Used internally by the VeriCall Edge system and should not be used by the user application.
VE_NAT_TYPE_OPEN	The VeriCall Edge system is not behind a NAT device.
VE_NAT_TYPE_FULL_CONE	The VeriCall Edge system is behind a full cone NAT device.
VE_NAT_TYPE_ADDRESS_RESTRICTED_CONE	The VeriCall Edge system is behind an address restricted cone NAT device.
VE_NAT_TYPE_PORT_RESTRICTED_CONE	The VeriCall Edge system is behind a port restricted cone NAT device.
VE_NAT_TYPE_SYMMETRIC_NAT	The VeriCall Edge system is behind a symmetric NAT device.
VE_NAT_TYPE_SYMMETRIC_FIREWALL	The VeriCall Edge system is behind a symmetric firewall device.
VE_NAT_TYPE_UDP_BLOCKED	The VeriCall Edge system is behind an UDP blocking NAT device.
VE_NUM_NAT_TYPES	Used internally by the VeriCall Edge system and should not be used by the user application.

B.230 t_ve_nat_request_type_e

The **t_ve_nat_request_type_e** enumeration is part of the **t_ve_user_msg_nat_request_failed_s** structure as described in Appendix B.33.3 on Page 361. The enumeration returns the NAT traversal module request type this event refers to. The **t_ve_nat_request_type_e** enumeration is defined in **ve_nat_defs.h** and is described below:

```
typedef enum{
  VE_NAT_REQUEST_TYPE_INVALID=0,
  VE_NAT_REQUEST_TYPE_VE_NATSTARTDISCOVER,
  VE_NAT_REQUEST_TYPE_VE_NATCANCELDISCOVER,
  VE_NAT_REQUEST_TYPE_VE_NATSETDISCOVERDATA,
  VE_NAT_REQUEST_TYPE_VE_NATRESERVEUSERADDRESSES,
  VE_NAT_REQUEST_TYPE_VE_NATRELEASEUSERADDRESSES,
  VE_NAT_REQUEST_TYPE_VE_NATSETCHANNELADDRESSESDATA,
  VE_NAT_REQUEST_TYPE_VE_NATSTARTKEEPALIVE,
  VE_NAT_REQUEST_TYPE_VE_NATSTOPKEEPALIVE,
  VE_NAT_NUM_REQUEST_TYPES
}t_ve_nat_request_type_e;
```

The enumeration elements are as follows:

Element	Definition
VE_NAT_REQUEST_TYPE_INVALID	Used internally by the VeriCall Edge system and should not be used by the user application.
VE_NAT_REQUEST_TYPE_VE_ NATSTARTDISCOVER	The event is a response to a **VE_NATStartDiscover** request.
VE_NAT_REQUEST_TYPE_VE_ NATCANCELDISCOVER	The event is a response to a **VE_NATCancelDiscover** request.
VE_NAT_REQUEST_TYPE_VE_ NATSETDISCOVERDATA	The event is a response to a **VE_NATSetDiscoverData** request.
VE_NAT_REQUEST_TYPE_VE_ NATRESERVEUSERADDRESSES	The event is a response to a **VE_NATReserveUserAddresses** request.
VE_NAT_REQUEST_TYPE_VE_ NATRELEASEUSERADDRESSES	The event is a response to a **VE_NATReleaseUserAddresses** request.
VE_NAT_REQUEST_TYPE_VE_ NATSETCHANNELADDRESSESDATA	The event is a response to a **VE_NATSetChannelAddressesData** request.
VE_NAT_REQUEST_TYPE_VE_ NATSTARTKEEPALIVE	The event is a response to a **VE_NATStartKeepAlive** request.
VE_NAT_REQUEST_TYPE_VE_ NATSTOPKEEPALIVE	The event is a response to a **VE_NATStopKeepAlive** request.
VE_NAT_NUM_REQUEST_TYPES	Used internally by the VeriCall Edge system and should not be used by the user application.

B.231 t_ve_nat_request_failed_reason_e

The **t_ve_nat_request_failed_reason_e** enumeration is part of the **t_ve_user_msg_nat_request_failed_s** structure as described in Appendix B.33.3 on Page 361. The enumeration returns the reason the event failure occurred. The **t_ve_nat_request_failed_reason_e** enumeration is defined in **ve_nat_defs.h** and is described below:

```
typedef enum{
  VE_NAT_REQUEST_FAILED_REASON_INVALID=0,
  VE_NAT_REQUEST_FAILED_INVALID_USER_HANDLE,
  VE_NAT_REQUEST_FAILED_BUSY,
  VE_NAT_REQUEST_FAILED_NO_FREE_JOB,
  VE_NAT_REQUEST_FAILED_TRANSPORT_ERROR,
  VE_NAT_REQUEST_FAILED_INVALID_PARAMETER,
  VE_NAT_REQUEST_FAILED_CANCELLED,
  VE_NAT_REQUEST_FAILED_STUN_ERROR,
  VE_NAT_REQUEST_FAILED_INTERNAL_ERROR,
  VE_NAT_REQUEST_FAILED_NOT_SUPPORTED,
  VE_NAT_NUM_REQUEST_FAILED_REASONS
}t_ve_nat_request_failed_reason_e;
```

The enumeration elements are as follows:

Element	Definition
VE_NAT_REQUEST_FAILED_REASON_INVALID	Used internally by the VeriCall Edge system and should not be used by the user application.
VE_NAT_REQUEST_FAILED_INVALID_USER_HANDLE	The failure was the result of an invalid user handle.
VE_NAT_REQUEST_FAILED_BUSY	The request cannot be performed while jobs are active.
VE_NAT_REQUEST_FAILED_NO_FREE_JOB	The failure was the result of no free jobs being available.
VE_NAT_REQUEST_FAILED_TRANSPORT_ERROR	The failure was the result of a transport (network) error.
VE_NAT_REQUEST_FAILED_INVALID_PARAMETER	The failure was the result of one or more invalid parameters.
VE_NAT_REQUEST_FAILED_CANCELLED	The failure was due to the request being cancelled before completion.
VE_NAT_REQUEST_FAILED_STUN_ERROR	The failure was the result of a STUN messaging error.
VE_NAT_REQUEST_FAILED_INTERNAL_ERROR	The failure was the result of an unspecified internal error.
VE_NAT_REQUEST_FAILED_NOT_SUPPORTED	The failure was due to the feature being unsupported. This may be due to the feature having been disabled in the associated configuration structures.
VE_NAT_NUM_REQUEST_FAILED_REASONS	Used internally by the VeriCall Edge system and should not be used by the user application.

B.232 t_ve_nat_keep_alive_event_e

The **t_ve_nat_keep_alive_event_e** enumeration is part of the **t_ve_user_msg_nat_keep_alive_event_s** structure as described in Appendix B.33.4 on Page 362. The enumeration returns the keep-alive feature that generated the event. The **t_ve_nat_keep_alive_event_e** enumeration is defined in **ve_nat_defs.h** and is described below:

```
typedef enum{
  VE_NAT_KEEP_ALIVE_EVENT_INVALID=0,
  VE_NAT_KEEP_ALIVE_EVENT_STARTED,
  VE_NAT_KEEP_ALIVE_EVENT_STARTED_DUP,
  VE_NAT_KEEP_ALIVE_EVENT_STOPPED,
  VE_NAT_KEEP_ALIVE_EVENT_STOPPED_DUP,
  VE_NAT_KEEP_ALIVE_EVENT_NO_RESPONSE,
  VE_NAT_KEEP_ALIVE_EVENT_NEW_BINDING,
  VE_NUM_NAT_KEEP_ALIVE_EVENTS
}t_ve_nat_keep_alive_event_e;
```

The enumeration elements are as follows:

Element	Definition
VE_NAT_KEEP_ALIVE_EVENT_INVALID	Used internally by the VeriCall Edge system and should not be used by the user application.
VE_NAT_KEEP_ALIVE_EVENT_STARTED	The event was raised by a keep-alive first response.
VE_NAT_KEEP_ALIVE_EVENT_STARTED_DUP	The event was raised by a duplicate keep-alive having been started.
VE_NAT_KEEP_ALIVE_EVENT_STOPPED	The event was raised by a keep-alive stopped feature.
VE_NAT_KEEP_ALIVE_EVENT_STOPPED_DUP	The event was raised by a duplicate keep-alive stopped feature.
VE_NAT_KEEP_ALIVE_EVENT_NO_RESPONSE	The event was raised by there being no response during the keep-alive period.
VE_NAT_KEEP_ALIVE_EVENT_NEW_BINDING	The event was raised by a keep-alive detected NAT binding change.
VE_NUM_NAT_KEEP_ALIVE_EVENTS	Used internally by the VeriCall Edge system and should not be used by the user application.

B.233 t_ve_user_msg_undecodeable_stream_reason_e

The **t_ve_user_msg_undecodeable_stream_reason_e** enumeration is part of the **t_ve_user_msg_undecodeable_stream_s** structure as described in Appendix B.33.67 on Page 422. The enumeration returns the reason the stream was un-decodeable. The **t_ve_user_msg_undecodeable_stream_reason_e** enumeration is defined in **ve_events.h** and is described below:

```
typedef enum{
  VE_UNDECODEABLE_STREAM_REASON_INVALID=0,
  VE_UNDECODEABLE_STREAM_REASON_UNKNOWN,
  VE_UNDECODEABLE_STREAM_REASON_H263_ANNEX_D_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_ANNEX_E_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_ANNEX_F_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_ANNEX_G_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_D_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_E_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_F_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_G_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_H_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_I_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_J_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_K_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_L_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_M_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_N_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_O_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_P_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_Q_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_R_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_S_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_H263_1998_ANNEX_T_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_MPEG4_VOS_HEADER_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_MPEG4_VO_HEADER_ERROR,
  VE_UNDECODEABLE_STREAM_REASON_MPEG4_VOL_HEADER_ERROR,
  VE_NUM_UNDECODEABLE_STREAM_REASONS
}t_ve_user_msg_undecodeable_stream_reason_e;
```

The enumeration elements are as follows:

Element	Definition
VE_UNDECODEABLE_STREAM_REASON_ INVALID=0,	Used internally by the VeriCall Edge system and should not be used by the user application.
VE_UNDECODEABLE_STREAM_REASON_ UNKNOWN,	An error was detected by the reason is unknown.
VE_UNDECODEABLE_STREAM_REASON_ H263_ANNEX_D_ERROR,	The stream error was due to a H.263 (1996) Annex D error.
VE_UNDECODEABLE_STREAM_REASON_ H263_ANNEX_E_ERROR,	The stream error was due to a H.263 (1996) Annex E error.
VE_UNDECODEABLE_STREAM_REASON_ H263_ANNEX_F_ERROR,	The stream error was due to a H.263 (1996) Annex F error.

Element	Definition
VE_UNDECODEABLE_STREAM_REASON_ H263_ANNEX_G_ERROR,	The stream error was due to a H.263 (1996) Annex G error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_D_ERROR,	The stream error was due to a H.263 (1998) Annex D error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_E_ERROR,	The stream error was due to a H.263 (1998) Annex E error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_F_ERROR,	The stream error was due to a H.263 (1998) Annex F error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_G_ERROR,	The stream error was due to a H.263 (1998) Annex G error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_H_ERROR,	The stream error was due to a H.263 (1998) Annex H error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_I_ERROR,	The stream error was due to a H.263 (1998) Annex I error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_J_ERROR,	The stream error was due to a H.263 (1998) Annex J error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_K_ERROR,	The stream error was due to a H.263 (1998) Annex K error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_L_ERROR,	The stream error was due to a H.263 (1998) Annex L error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_M_ERROR,	The stream error was due to a H.263 (1998) Annex M error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_N_ERROR,	The stream error was due to a H.263 (1998) Annex N error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_O_ERROR,	The stream error was due to a H.263 (1998) Annex O error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_P_ERROR,	The stream error was due to a H.263 (1998) Annex P error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_Q_ERROR,	The stream error was due to a H.263 (1998) Annex Q error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_R_ERROR,	The stream error was due to a H.263 (1998) Annex R error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_S_ERROR,	The stream error was due to a H.263 (1998) Annex S error.
VE_UNDECODEABLE_STREAM_REASON_ H263_1998_ANNEX_T_ERROR,	The stream error was due to a H.263 (1998) Annex T error.
VE_UNDECODEABLE_STREAM_REASON_ MPEG4_VOS_HEADER_ERROR,	The stream error was due to a MPEG-4 VOS header error.
VE_UNDECODEABLE_STREAM_REASON_ MPEG4_VO_HEADER_ERROR,	The stream error was due to a MPEG-4 VO header error.
VE_UNDECODEABLE_STREAM_REASON_ MPEG4_VOL_HEADER_ERROR,	The stream error was due to a MPEG-4 VOL header error.
VE_NUM_UNDECODEABLE_STREAM_REASONS	Used internally by the VeriCall Edge system and should not be used by the user application.

Appendix C

Attribute to Structure Association

This Appendix lists the association between the VeriCall Edge attribute enumeration types, defined in Appendix B.71 on Page 504, and the structures that are used by **VE_SetSystemAttribute** and **VE_GetSystemAttribute** to read or alter those VeriCall Edge attributes.

C.1 Attribute/Structure Association

Table C.1 below provides a complete list of the VeriCall Edge attributes and the associated structure used to read or alter those attributes.

Table C.1: VeriCall Edge Attribute to Structure Association

Attribute	Structure Association
VE_ATTRIBUTE_TYPE_ NO_VALUE	Not accessible.
VE_ATTRIBUTE_TYPE_ INITIALIZATION_ATTRIBUTES	**t_ve_init_config_s** structure, described in Appendix B.2 on Page 254.
VE_ATTRIBUTE_TYPE_ DEFAULT_IP	**t_ve_ip_default_configuration_s** structure, described in Appendix B.7 on Page 260
VE_ATTRIBUTE_TYPE_ DEFAULT_IP_SETTINGS	**t_ve_ip_settings_default_configuration_s** structure, described in Appendix B.8 on Page 261.
VE_ATTRIBUTE_TYPE_ DEFAULT_SYSTEM_IP_SETTINGS	**t_ve_ip_settings_default_configuration_s** structure, described in Appendix B.8 on Page 261.
VE_ATTRIBUTE_TYPE_ DEFAULT_SIP_IP_SETTINGS	**t_ve_ip_settings_default_configuration_s** structure, described in Appendix B.8 on Page 261.
VE_ATTRIBUTE_TYPE_ DEFAULT_H323_IP_SETTINGS	**t_ve_ip_settings_default_configuration_s** structure, described in Appendix B.8 on Page 261.
VE_ATTRIBUTE_TYPE_ DEFAULT_NAT_IP_SETTINGS	**t_ve_ip_settings_default_configuration_s** structure, described in Appendix B.8 on Page 261.
VE_ATTRIBUTE_TYPE_ DEFAULT_NET_IP_SETTINGS	**t_ve_ip_settings_default_configuration_s** structure, described in Appendix B.8 on Page 261.
VE_ATTRIBUTE_TYPE_ DEFAULT_SIP_CONFIGURATION	**t_ve_sip_default_configuration_s** defined in Appendix B.17 on Page 284.

Attribute	Structure Association
VE_ATTRIBUTE_TYPE_ MEDIA_PROFILE	**t_ve_media_definition_s** defined in Appendix B.29 on Page 315.
VE_ATTRIBUTE_TYPE_ DEFAULT_AJBM	**t_ve_ajbm_algorithm_default_configuration_s** defined in Appendix B.9 on Page 262.
VE_ATTRIBUTE_TYPE_ DEFAULT_PRE_PROC_CONFIGURATION	**t_ve_pre_proc_default_configuration_s** defined in Appendix B.15 on Page 275.
VE_ATTRIBUTE_TYPE_ DEFAULT_POST_PROC_CONFIGURATION	**t_ve_post_proc_default_configuration_s** defined in Appendix B.16 on Page 280.
VE_ATTRIBUTE_TYPE_ DEFAULT_NTG	**t_ve_ntg_algorithm_default_configuration_s** defined in Appendix B.14 on Page 275.
VE_ATTRIBUTE_TYPE_ DEFAULT_NTD	**t_ve_ntd_algorithm_default_configuration_s** defined in Appendix B.13 on Page 274.
VE_ATTRIBUTE_TYPE_ DEFAULT_LINE_ECAN	**t_ve_line_ecan_configuration_s** defined in Appendix B.10 on Page 264.
VE_ATTRIBUTE_TYPE_ CHANNEL_CONFIGURATION	**t_ve_channel_s** defined in Appendix B.21 on Page 289.
VE_ATTRIBUTE_TYPE_ USER_TRANSPORT_OPEN_FUNCTION	No associated structure with this attribute, it is an **UWord32** element.
VE_ATTRIBUTE_TYPE_ USER_TRANSPORT_SEND_FUNCTION	No associated structure with this attribute, it is an **UWord32** element.
VE_ATTRIBUTE_TYPE_ USER_TRANSPORT_RECEIVE_FUNCTION	No associated structure with this attribute, it is an **UWord32** element.
VE_ATTRIBUTE_TYPE_ USER_TRANSPORT_CLOSE_FUNCTION	No associated structure with this attribute, it is an **UWord32** element.
VE_ATTRIBUTE_TYPE_ USER_TRANSPORT_CONFIGURE_FUNCTION	No associated structure with this attribute, it is an **UWord32** element.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_TRANSFER_OPEN_FUNCTION	No associated structure with this attribute, it is an **UWord32** element.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_DATA_TRANSFER_FUNCTION	No associated structure with this attribute, it is an **UWord32** element.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_TRANSFER_CLOSE_FUNCTION	No associated structure with this attribute, it is an **UWord32** element.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_TRANSFER_AVAIL_FUNCTION	No associated structure with this attribute, it is an **UWord32** element.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_TRANSFER_QUERY_FUNCTION	No associated structure with this attribute, it is an **UWord32** element.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_TRANSFER_CONFIG_FUNCTION	No associated structure with this attribute, it is an **UWord32** element.
VE_ATTRIBUTE_TYPE_ READONLY_COMMIT_LOCK	No associated structure with this attribute, it is an **UWord32** element.
VE_ATTRIBUTE_TYPE_ READONLY_INITIALIZE_LOCK	No associated structure with this attribute, it is an **UWord32** element.
VE_ATTRIBUTE_TYPE_ READONLY_USER	**t_ve_user_channel_details_s** defined in Appendix B.26 on Page 300.
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_SYSTEM_CALLS	**t_ve_stat_system_calls_s** defined in Appendix B.38.1 on Page 432.

Attribute	Structure Association
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_VIDEO_ENCODER	**t_ve_stat_channel_video_encoder_s** defined in Appendix B.38.3 on Page 433.
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_VIDEO_DECODER	**t_ve_stat_channel_video_decoder_s** defined in Appendix B.38.4 on Page 434.
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_RTCP	**t_ve_stat_channel_rtcp_s** defined in Appendix B.38.5 on Page 435.
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_RTCP_PEER	**t_ve_stat_channel_rtcp_peer_s** defined in Appendix B.38.6 on Page 438.
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_AV_SYNC	**t_ve_stat_av_sync_s** defined in Appendix B.38.2.1 on Page 433.
VE_ATTRIBUTE_TYPE_ READONLY_USER_SIP_DETAILS	**t_ve_sip_user_registration_s** defined in Appendix B.39 on Page 440.
VE_ATTRIBUTE_TYPE_READONLY_ DEFAULT_STREAM_STRUCTURE	**t_ve_stream_configuration_s** defined in Appendix B.24 on Page 298.
VE_ATTRIBUTE_TYPE_READONLY_ LAST_START_STREAM_STRUCTURE	**t_ve_last_stream_configuration_s** defined in Appendix B.5 on Page 257.
VE_ATTRIBUTE_TYPE_ DEFAULT_DATA_END_POINT	**t_ve_data_end_point_default_configuration_s** defined in Appendix B.19 on Page 287.
VE_ATTRIBUTE_TYPE_ DEFAULT_TX_LIP_SYNC	**t_ve_tx_lip_sync_default_configuration_s** defined in Appendix B.18 on Page 287.
VE_ATTRIBUTE_TYPE_ READONLY_USER_H323_DETAILS	**t_ve_h323_user_registration_s** defined in Appendix B.52 on Page 467.
VE_ATTRIBUTE_TYPE_ DEFAULT_H323_CONFIGURATION	**t_ve_h323_default_configuration_s** defined in Appendix B.58 on Page 475.
VE_ATTRIBUTE_TYPE_ SYSTEM_LOGGING	**t_ve_system_logging_s** defined in Appendix B.4 on Page 256.
VE_ATTRIBUTE_TYPE_ DEFAULT_ACOUSTIC_ECAN	**t_ve_acoustic_ecan_configuration_s** defined in Appendix B.11 on Page 266.
VE_ATTRIBUTE_TYPE_ VIDEO_TEAR_PROTECT	**t_ve_video_tear_protect_default_configuration_s** defined in Appendix B.20 on Page 288.
VE_ATTRIBUTE_TYPE_ DEFAULT_NAT_CONFIGURATION	**t_ve_nat_default_configuration_s** defined in Appendix B.60 on Page 491.
VE_ATTRIBUTE_TYPE_READONLY_ USER_SIP_PRESENCE_DETAILS	**t_ve_sip_presence_data_s** defined in Appendix B.46 on Page 452.
VE_ATTRIBUTE_TYPE_READONLY_ USER_SIP_MW_DETAILS	**t_ve_sip_mw_data_s** defined in Appendix B.47 on Page 457.
VE_ATTRIBUTE_TYPE_ DEFAULT_NET_CONFIGURATION	**t_ve_net_default_configuration_s** defined in Appendix B.65 on Page 499.
VE_ATTRIBUTE_TYPE_ DEFAULT_GAIN_CONTROL_TX	t_ve_gain_control_configuration_s structure defined in Appendix B.12 on Page 273
VE_ATTRIBUTE_TYPE_ DEFAULT_GAIN_CONTROL_RX	t_ve_gain_control_configuration_s structure defined in Appendix B.12 on Page 273
VE_ATTRIBUTE_TYPE_ NUMBER_OF_ATTRIBUTES	Not accessible.

C.2 Attribute/Index Association

Table C.2 below provides a complete list of the VeriCall Edge attributes and the associated Attribute Index values used to read or alter those attributes.

Table C.2: VeriCall Edge Attribute/Index Structure Association

Attribute Type	Attribute Index
VE_ATTRIBUTE_TYPE_ NO_VALUE	Not accessible.
VE_ATTRIBUTE_TYPE_ INITIALIZATION_ATTRIBUTES	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ DEFAULT_IP	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ DEFAULT_IP_SETTINGS	Assuming that the **VE_UpdateSystemAttribute** function has not been used to update this attribute type, the attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ DEFAULT_SYSTEM_IP_SETTINGS	Attribute Index should be of type Channel Number and should not exceed that defined in **t_ve_init_config_s** described in Appendix B.2 on Page 254. Note that channel numbers start from one.
VE_ATTRIBUTE_TYPE_ DEFAULT_SIP_IP_SETTINGS	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ DEFAULT_H323_IP_SETTINGS	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ DEFAULT_NAT_IP_SETTINGS	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ DEFAULT_NET_IP_SETTINGS	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ DEFAULT_SIP_CONFIGURATION	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ MEDIA_PROFILE	Attribute Index should be of type Media ID as defined in Appendix B.110 on Page 539.
VE_ATTRIBUTE_TYPE_ DEFAULT_AJBM	Attribute Index should be of type Channel Number and should not exceed that defined in **t_ve_init_config_s** described in Appendix B.2 on Page 254. Note that channel numbers start from one.
VE_ATTRIBUTE_TYPE_ DEFAULT_PRE_PROC_CONFIGURATION	Attribute Index should be of type Channel Number and should not exceed that defined in **t_ve_init_config_s** described in Appendix B.2 on Page 254. Note that channel numbers start from one.
VE_ATTRIBUTE_TYPE_ DEFAULT_POST_PROC_CONFIGURATION	Attribute Index should be of type Channel Number and should not exceed that defined in **t_ve_init_config_s** described in Appendix B.2 on Page 254. Note that channel numbers start from one.
VE_ATTRIBUTE_TYPE_ DEFAULT_NTG	Attribute Index should be of type Channel Number and should not exceed that defined in **t_ve_init_config_s** described in Appendix B.2 on Page 254. Note that channel numbers start from one.
VE_ATTRIBUTE_TYPE_ DEFAULT_NTD	Attribute Index should be of type Channel Number and should not exceed that defined in **t_ve_init_config_s** described in Appendix B.2 on Page 254. Note that channel numbers start from one.

Attribute Type	Attribute Index
VE_ATTRIBUTE_TYPE_ DEFAULT_LINE_ECAN	Attribute Index should be of type Channel Number and should not exceed that defined in **t_ve_init_config_s** described in Appendix B.2 on Page 254. Note that channel numbers start from one.
VE_ATTRIBUTE_TYPE_ CHANNEL_CONFIGURATION	Attribute Index should be of type Channel Number and should not exceed that defined in **t_ve_init_config_s** described in Appendix B.2 on Page 254. Note that channel numbers start from one.
VE_ATTRIBUTE_TYPE_ USER_TRANSPORT_OPEN_FUNCTION	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ USER_TRANSPORT_SEND_FUNCTION	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ USER_TRANSPORT_RECEIVE_FUNCTION	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ USER_TRANSPORT_CLOSE_FUNCTION	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ USER_TRANSPORT_CONFIGURE_FUNCTION	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_TRANSFER_OPEN_FUNCTION	Attribute Index should be of type *identifier* specified when the user function was registered with the VeriCall Edge system.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_DATA_TRANSFER_FUNCTION	Attribute Index should be of type *identifier* specified when the user function was registered with the VeriCall Edge system.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_TRANSFER_CLOSE_FUNCTION	Attribute Index should be of type *identifier* specified when the user function was registered with the VeriCall Edge system.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_TRANSFER_AVAIL_FUNCTION	Attribute Index should be of type *identifier* specified when the user function was registered with the VeriCall Edge system.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_TRANSFER_QUERY_FUNCTION	Attribute Index should be of type *identifier* specified when the user function was registered with the VeriCall Edge system.
VE_ATTRIBUTE_TYPE_USER_DATA_ CHANNEL_TRANSFER_CONFIG_FUNCTION	Attribute Index should be of type *identifier* specified when the user function was registered with the VeriCall Edge system.
VE_ATTRIBUTE_TYPE_ READONLY_COMMIT_LOCK	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ READONLY_INITIALIZE_LOCK	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ READONLY_USER	Attribute Index should be of type User.
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_SYSTEM_CALLS	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_VIDEO_ENCODER	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_VIDEO_DECODER	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_RTCP	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_RTCP_PEER	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_READONLY_ STATISTICS_AV_SYNC	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.

Attribute Type	Attribute Index
VE_ATTRIBUTE_TYPE_ READONLY_USER_SIP_DETAILS	Attribute Index should be of type User Handle and should not exceed that defined in **t_ve_init_config_s** described in Appendix B.2 on Page 254. Note that channel numbers start from one.
VE_ATTRIBUTE_TYPE_READONLY_ DEFAULT_STREAM_STRUCTURE	Attribute Index should be of type User.
VE_ATTRIBUTE_TYPE_READONLY_ LAST_START_STREAM_STRUCTURE	Attribute Index should be of type User.
VE_ATTRIBUTE_TYPE_ DEFAULT_DATA_END_POINT	Attribute Index should be of type *label*, which should be a unique arbitrary UWord32. This label should be used in identifying this data-end point instance.
VE_ATTRIBUTE_TYPE_ DEFAULT_TX_LIP_SYNC	Attribute Index should be of type Channel Number and should not exceed that defined in **t_ve_init_config_s** described in Appendix B.2 on Page 254. Note that channel numbers start from one.
VE_ATTRIBUTE_TYPE_ READONLY_USER_H323_DETAILS	Attribute Index should be of type User Handle and should not exceed that defined in **t_ve_init_config_s** described in Appendix B.2 on Page 254. Note that channel numbers start from one.
VE_ATTRIBUTE_TYPE_ DEFAULT_H323_CONFIGURATION	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ SYSTEM_LOGGING	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ DEFAULT_ACOUSTIC_ECAN	Attribute Index should be of type Channel Number and should not exceed that defined in **t_ve_init_config_s** described in Appendix B.2 on Page 254. Note that channel numbers start from one.
VE_ATTRIBUTE_TYPE_ VIDEO_TEAR_PROTECT	Attribute Index should be of type Channel Number and should not exceed that defined in **t_ve_init_config_s** described in Appendix B.2 on Page 254. Note that channel numbers start from one.
VE_ATTRIBUTE_TYPE_ DEFAULT_NAT_CONFIGURATION	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_READONLY_ USER_SIP_PRESENCE_DETAILS	Attribute Index should be of type User.
VE_ATTRIBUTE_TYPE_READONLY_ USER_SIP_MW_DETAILS	Attribute Index should be of type User.
VE_ATTRIBUTE_TYPE_ DEFAULT_NET_CONFIGURATION	Attribute Index should be of type VE_SINGLETON_DATA_ENTITY.
VE_ATTRIBUTE_TYPE_ DEFAULT_GAIN_CONTROL_TX	Attribute Index should be of type Channel Number and should not exceed that defined in **t_ve_init_config_s** described in Appendix B.2 on Page 254. Note that channel numbers start from one.
VE_ATTRIBUTE_TYPE_ DEFAULT_GAIN_CONTROL_RX	Attribute Index should be of type Channel Number and should not exceed that defined in **t_ve_init_config_s** described in Appendix B.2 on Page 254. Note that channel numbers start from one.
VE_ATTRIBUTE_TYPE_ NUMBER_OF_ATTRIBUTES	Not accessible.

Appendix D

Header Files & Type Definitions

This Appendix lists the VeriCall Edge header files and type definitions used.

- Section D.1 on Page 666 Lists the public VeriCall Edge header files.
- Section D.2 on Page 668 Details the VeriCall Edge data-type definitions.
- Section D.3 on Page 668 Lists Macro definitions.

D.1 Header Files

The following header files are required when using the VeriCall Edge API functions:

- **ve_alarms.h**
- **ve_callback.h**
- **ve_channel.h**
- **ve_configuration.h**
- **ve_control_audio.h**
- **ve_control_network.h**
- **ve_control_video.h**
- **ve_defs.h**
- **ve_eapi.h**
- **ve_error.h**
- **ve_events.h**
- **ve_media.h**
- **ve_prototypes.h**
- **ve_tones.h**
- **ve_types.h**

- **ve_h323_alarms.h**
- **ve_h323_configuration.h**
- **ve_h323_defs.h**
- **ve_h323_events.h**
- **ve_h323_prototypes.h**

- **ve_nat_configuration.h**
- **ve_nat_defs.h**
- **ve_nat_events.h**
- **ve_nat_prototypes.h**

Continued >>

- **ve_net_configuration.h**
- **ve_net_defs.h**
- **ve_net_events.h**
- **ve_net_prototypes.h**

- **ve_sip_alarms.h**
- **ve_sip_configuration.h**
- **ve_sip_defs.h**
- **ve_sip_events.h**
- **ve_sip_prototypes.h**

D.2 Type Definitions

The VeriCall Edge functions and structures described in this manual use a set of proprietary data-type definitions, defined in the file **ve_types.h**. These are as follows:

Type	Definition
Word8	Signed 8-bit character – single byte.
UWord8	Unsigned 8-bit character – single byte.
Word16	Signed 16-bit short integer.
UWord16	Unsigned 16-bit short integer.
Word32	Signed 32-bit long integer.
UWord32	Unsigned 32-bit long integer.

D.3 Macro Definitions

The following macros are provided in the **ve_defs.h** header file for use within the user application.

`VE_SINGLETON_DATA_ENTITY`

A macro that provides an argument for all singleton data entities (data entities for which there is, and only ever will be, a single entry). Designed specifically for use as the *`ve_attribute_index`* argument for all singleton data entities in **VE_SetSystemAttribute**, **VE_GetSystemAttribute** and **VE_RemoveSystemAttribute** function calls. The macro is defined as follows:

```
#define VE_SINGLETON_DATA_ENTITY 1
```

Appendix E

VeriCall Edge™ Terminology

This appendix describes terms used throughout the VeriCall Edge documentation set which you may find useful to refer to. Note that this is not a telephony terminology reference.

Term	Definition
AEC	Acoustic Echo Canceller
Adaptive Jitter Buffer	Buffer used to remove jitter, the variation in packet arrival time (AJBM).
API	Application Programming Interface.
Channel	A channel is a path through the VeriCall Edge system from the media to packet side that is constructed from a series of attributes that define the channels algorithmic capabilities.
CNG	Comfort Noise Generation – software that may be used by a voice codec to maintain an acceptable level of background noise during periods when there is no voice activity.
Codec	An encoder/decoder algorithm.
DTMF	Dual Tone Multi Frequency.
ECAN	Echo Canceller – applicable to an Acoustic or Line Echo Canceller.
GSM	Global System for Mobile communications.
MW	Message Waiting – mailbox system used to store end user messages , normally on a remote server.
MWI	Message Waiting Indicator - an application specific event to notify the end user of a message waiting event.
RTP	Real Time Transport Protocol – protocol used for transport across a packet network. Normally used with UDP.
RTCP	Real Time Control Protocol – used in accordance with RTP.
Session	A simple two-way communication pipe, it could be a simple telephone call or a multi-media session.
SIP	Session Initiation Protocol - a signaling protocol used for establishing sessions in an IP network.
Stream	A stream is a media or packet stream entering or leaving the VeriCall Edge system.
URI	Uniform Resource Identifier - used to identify a single endpoint, in the case of SIP generally a user identified by an address.
VAD	Voice Activity Detection (VAD) – software that may be used by an algorithm to reduce use of bandwidth by determining whether speech is present.
V2IP	Video and Voice Over IP.

Index

Reference **Page**

A

API
- Data Manager 77
- H.323 181
- Media Control 49
- NAT 225
- SIP 103
- System 7
- User 39

Attribute to Structure Association 659

D

Data Manager API
- Functions 78
- VE_RegisterChannelDataDeviceConfigure 100
- VE_RegisterChannelDataDeviceQuery 97
- VE_RegisterChannelDataSourceSinkClose 93
- VE_RegisterChannelDataSourceSinkOpen 89
- VE_RegisterChannelDataTransfer 91
- VE_RegisterChannelDataTransferAvailability . 95
- VE_RegisterPacketTransportClose 87
- VE_RegisterPacketTransportConfigure 85
- VE_RegisterPacketTransportOpen 79
- VE_RegisterPacketTransportReceive 83
- VE_RegisterPacketTransportSend 81

E

Enumerations
- t_ve_acoustic_ecan_aes_mode_e 525
- t_ve_acoustic_ecan_id_e 524
- t_ve_ajbm_algorithm_default_lip_sync_state_e 527
- t_ve_alarm_h323_subtype_e 583
- t_ve_alarm_sip_subtype_e 582

Reference **Page**
- t_ve_alarm_system_subtype_e 581
- t_ve_alarm_type_e 581
- t_ve_attribute_type_e 504
- t_ve_audio_control_data_audio_mute_direction_e 573
- t_ve_audio_control_data_audio_mute_state_e 572
- t_ve_audio_control_play_data_format_e 571
- t_ve_audio_control_play_direction_e 570
- t_ve_audio_control_record_data_format_e 570
- t_ve_audio_control_type_e 568
- t_ve_callback_audio_endian_e 596
- t_ve_callback_data_channel_media_format_e 597
- t_ve_callback_direction_e 596
- t_ve_callback_duplex_capability_e 598
- t_ve_callback_protocol_e 595
- t_ve_callback_transfermode_e 599
- t_ve_callback_transport_type_e 595
- t_ve_callback_video_color_format_e 598
- t_ve_callback_video_frame_buffer_e 599
- t_ve_channel_config_boolean_e 534
- t_ve_channel_data_format_e 534
- t_ve_channel_type_config_e 535
- t_ve_country_e 519
- t_ve_data_end_point_default_type_e 533
- t_ve_default_configuration_boolean_e 521
- t_ve_encode_image_event_type_e 650
- t_ve_g711_default_mode_e 522
- t_ve_gain_control_agc_mode_e 528
- t_ve_gain_control_mode_e 528
- t_ve_h323_call_forward_reason_e 625
- t_ve_h323_call_hold_response_e 632
- t_ve_h323_call_response_e 626, 634
- t_ve_h323_call_start_method_e 646
- t_ve_h323_call_waiting_flag_e 642
- t_ve_h323_forward_response_e 636
- t_ve_h323_gatekeeper_discovery_e 644
- t_ve_h323_hold_method_e 647
- t_ve_h323_mwi_response_e 633

t_ve_h323_name_id_char_set_e 640
t_ve_h323_name_id_restriction_e..... 641
t_ve_h323_q931_annex_e_protocol_preference_e 643
t_ve_h323_reg_response_e 629
t_ve_h323_request_failed_reason_e 639
t_ve_h323_request_type_e..... 637
t_ve_h323_user_sub_channel_select_e..... 648
t_ve_h323_video_feature_request_e 645
t_ve_high_pass_filter_state_e 535
t_ve_ip_default_configuration_e..... 520
t_ve_ip_port_reopen_e..... 520
t_ve_last_stream_state_e..... 558
t_ve_line_16ms_ecan_nlp_default_settings_e 523
t_ve_line_ecan_id_e..... 526
t_ve_media_data_type_e 536
t_ve_media_eg711_codec_and_law_e 541
t_ve_media_g711_mode_e..... 538
t_ve_media_g723_1_hpf_enable_boolean_e... 551
t_ve_media_g723_1_mode_e..... 551
t_ve_media_g723_1_pf_enable_boolean_e..... 552
t_ve_media_g723_1_rate_e..... 550
t_ve_media_gsm_amr_mode_e..... 554
t_ve_media_gsm_amr_rate_e..... 553
t_ve_media_gsm_amr_vad_mode_e 555
t_ve_media_gsm_amr_vad_state_e..... 556
t_ve_media_id_e 539
t_ve_media_ip_configuration_e 537
t_ve_media_stream_direction_e..... 557
t_ve_media_transport_type_e..... 536
t_ve_media_type_e..... 537
t_ve_media_vad_enable_boolean_e..... 538
t_ve_media_vad_state_e 549
t_ve_media_video_config_state_e 546
t_ve_media_video_encoding_mode_e 542
t_ve_media_video_encoding_mode_rate_e..... 543
t_ve_media_video_frame_type_selection_e.... 542
t_ve_media_video_profile_level_e 548
t_ve_media_video_profile_selection_e 543
t_ve_media_video_quantization_param_e 544
t_ve_media_video_simple_profile_e..... 547
t_ve_media_video_stream_type_e 549
t_ve_module_class_e 559
t_ve_module_state_e..... 559
t_ve_nat_discover_method_e..... 650
t_ve_nat_keep_alive_event_e..... 655
t_ve_nat_request_failed_reason_e..... 654
t_ve_nat_request_type_e..... 653
t_ve_nat_state_e..... 651
t_ve_nat_types_e..... 652
t_ve_net_lookup_record_type_e 566
t_ve_net_lookup_type_e 563
t_ve_net_request_failed_reason_e..... 567
t_ve_net_request_type_e..... 566
t_ve_net_service_protocol_e..... 565
t_ve_net_service_type_e 564
t_ve_network_connection_event_e 562
t_ve_network_connection_type_e 561
t_ve_network_control_type_e 560
t_ve_ntd_country_type_default_configuration_e 529
t_ve_process_selector_default_configuration_e 521
t_ve_return_code_e 510
t_ve_signaling_type_e..... 515
t_ve_sip_authentication_required_e 608
t_ve_sip_call_hold_method_e 609
t_ve_sip_call_hold_state_e..... 609
t_ve_sip_date_state_e..... 607
t_ve_sip_deregister_contacts_flag_e 606
t_ve_sip_dialog_direction_e..... 622
t_ve_sip_dialog_field_state_e 619
t_ve_sip_dialog_state_e..... 620
t_ve_sip_dialog_state_event_e 621
t_ve_sip_mw_indicatorstatus_e..... 617
t_ve_sip_mw_message_context_e..... 618
t_ve_sip_mw_start_flag_e..... 616
t_ve_sip_presence_type_e..... 615
t_ve_sip_presence_xml_pidf_status_basic_type_e 610
t_ve_sip_privacy_e..... 624
t_ve_sip_request_failed_reason_e..... 591
t_ve_sip_request_type_e 611
t_ve_sip_require_prack_e..... 606
t_ve_sip_response_code_e 602
t_ve_sip_rx_session_response_content_e..... 608
t_ve_sip_state_flag_default_configuration_e .. 605
t_ve_sip_subscribe_state_e 614
t_ve_sip_symmetric_mode_default_configuration_e..... 604
t_ve_sip_transport_protocol_e 623
t_ve_sip_tx_session_response_content_e..... 607
t_ve_stat_av_sync_correlate_e..... 601
t_ve_stats_type_e 600
t_ve_system_logging_destination_e..... 514
t_ve_system_logging_method_e 515
t_ve_tone_direction_e 518
t_ve_tone_transmission_media_type_e 516
t_ve_tone_type_e..... 517
t_ve_user_callback_return_code_e..... 594
t_ve_user_message_type_e 584
t_ve_user_msg_encode_image_event_e..... 649
t_ve_user_msg_module_state_transition_reason_e..... 590
t_ve_user_msg_play_prompt_event_e 648
t_ve_user_msg_record_prompt_event_e 649
t_ve_user_msg_stream_event_reason_e..... 589
t_ve_user_msg_stream_event_type_e 590

t_ve_user_msg_undecodeable_stream_reason_e 656
t_ve_user_sub_channel_index_e 594
t_ve_user_sub_channel_select_e 593
t_ve_verbose_level_e 513
t_ve_video_color_format_e 532
t_ve_video_config_error_reason_e 580
t_ve_video_control_encode_format_e 579
t_ve_video_control_filter_selector_e 576
t_ve_video_control_image_freeze_state_e 579
t_ve_video_control_mirror_e 577
t_ve_video_control_mute_mode_e 578
t_ve_video_control_rotate_e 577
t_ve_video_control_select_display_state_e 576
t_ve_video_control_type_e 574
t_ve_video_default_image_mirror_e 531
t_ve_video_default_image_rotate_e 530
t_ve_video_default_local_image_enable_e 530
t_ve_video_filter_selection_e 532
Error Codes 245

H

H.323 API
Functions 182
H.323 API
VE_H323RegisterUser 183
H.323 API
VE_H323DeregisterUser 185
H.323 API
VE_H323AddUsersRegistrationData 187
H.323 API
VE_H323UpdateUsersRegistrationData 189
H.323 API
VE_H323RemoveUsersRegistrationData 191
H.323 API
VE_H323AddUsersRegistrationData 193
H.323 API
VE_H323StartCall 195
H.323 API
VE_H323StopCall 197
H.323 API
VE_H323UpdateCall 199
H.323 API
VE_H323CallResponse 201
H.323 API
VE_H323SendUserInput 203
H.323 API
VE_H323VideoFastUpdate 205
H.323 API
VE_H323VideoFreeze 207
H.450 API
VE_H323CallForward 213
VE_H323CallForwardActivate 215
VE_H323CallForwardDeactivate 217
VE_H323CallHoldOff 221
VE_H323CallHoldOn 219
VE_H323CallTransferConsult 209
VE_H323CallTransferDontConsult 211
VE_H323MWIInterrogate 223
Header Files 666

M

Macro Definitions 668
Media Control API
Functions 50
VE_ControlAudio 69
VE_ControlNetwork 73
VE_ControlVideo 71
VE_GenerateDigitString 66
VE_GenerateNamedTone 58
VE_GenerateUserTone 62
VE_NETLookUp 75
VE_StartStream 51
VE_StopStream 53
VE_UpdateStream 55

N

NAT API
Functions 226
VE_NATCancelDiscover 231
VE_NATGetChannelAddressesData 236
VE_NATGetDiscoverData 230
VE_NATReleaseUserAddresses 238
VE_NATReserveUserAddresses 232
VE_NATSetChannelAddressesData 234
VE_NATSetDiscoverData 229
VE_NATStartDiscover 227
VE_NATStartKeepAlive 240
VE_NATStopKeepAlive 242

R

Return Codes 245

S

SIP API
Functions 104
VE_SIPAddUsersPresenceData 158
VE_SIPAddUsersRegistrationData 110
VE_SIPDeregisterUser 108
VE_SIPGetUsersPresenceData 160
VE_SIPGetUsersRegistrationData 112
VE_SIPInterrogateDialogSubscription 179
VE_SIPInterrogateMessageWaiting 173

VE_SIPQueryRegistration 120
VE_SIPRegisterUser 106
VE_SIPRemoveUsersPresenceData 164
VE_SIPRemoveUsersRegistrationData 116
VE_SIPSendInfo 153
VE_SIPSendMessage 147
VE_SIPSendOptions 150
VE_SIPSendService 156
VE_SIPSessionHoldOff 138
VE_SIPSessionHoldOn 136
VE_SIPSessionReinvite 134
VE_SIPSessionResponse 129
VE_SIPSessionStart 122
VE_SIPSessionStart_WithConfig 124
VE_SIPSessionStop 127
VE_SIPSessionTransfer 140
VE_SIPSessionTransferConsult 143
VE_SIPSessionTransferDontConsult 145
VE_SIPSessionUpdate 131
VE_SIPStartDialogSubscription 175
VE_SIPStartMessageWaitingSubscription 169
VE_SIPStartPresenceSubscription 165
VE_SIPStopDialogSubscription 177
VE_SIPStopMessageWaitingSubscription 171
VE_SIPStopPresenceSubscription 167
VE_SIPUpdateRegistration 118
VE_SIPUpdateUsersPresenceData 162
VE_SIPUpdateUsersRegistrationData 114

Structures

t_ve_acoustic_ecan_aec_complex_configuration_s 268
t_ve_acoustic_ecan_aec_configuration_s 267
t_ve_acoustic_ecan_aes_configuration_s 271
t_ve_acoustic_ecan_configuration_s 266
t_ve_ajbm_algorithm_default_configuration_s 262
t_ve_ajbm_algorithm_default_lip_sync_config_s 263
t_ve_alarm_subtype_ajbm_discard_buffer_full_s 414
t_ve_alarm_subtype_ajbm_extended_break_in_stream_s 414
t_ve_alarm_subtype_ajbm_underrun_s 415
t_ve_alarm_subtype_h323_alarm_s 416
t_ve_alarm_subtype_sip_alarm_s 416
t_ve_audio_control_data_audio_mute_s 346
t_ve_audio_control_data_end_point_selector_s 345
t_ve_audio_control_lip_sync_adjustment_s 345
t_ve_audio_control_play_start_s 344
t_ve_audio_control_record_start_s 344
t_ve_audio_control_s 343
t_ve_audio_module_state_s 259
t_ve_audio_video_module_state_s 259
t_ve_callback_audio_data_channel_config_s . 296
t_ve_callback_audio_device_query_s 427
t_ve_callback_channel_data_configuration_u . 295
t_ve_callback_data_channel_descriptor_s 424
t_ve_callback_device_query_u 426
t_ve_callback_host_descriptor_s 425
t_ve_callback_network_config_s 430
t_ve_callback_network_open_s 297
t_ve_callback_video_capture_device_query_s 429
t_ve_callback_video_data_channel_config_s .. 296
t_ve_callback_video_display_device_query_s 428
t_ve_channel_3_way_calling_config_s 290
t_ve_channel_data_configuration_s 291
t_ve_channel_packet_configuration_s 293
t_ve_channel_port_configuration_s 294
t_ve_channel_post_conditioning_s 292
t_ve_channel_pre_conditioning_s 292
t_ve_channel_s 289
t_ve_data_end_point_default_configuration_s 287
t_ve_g711_post_proc_default_configuration_s 281
t_ve_g711_pre_proc_default_configuration_s 277
t_ve_gain_control_configuration_s 273
t_ve_h323_address_s 469
t_ve_h323_call_handle_s 469
t_ve_h323_call_update_attributes_s 473
t_ve_h323_call_update_user_channel_association_s_s 471
t_ve_h323_call_update_user_channel_details_s 470
t_ve_h323_contact_info_s 468
t_ve_h323_default_configuration_s 475
t_ve_h323_gatekeeper_s 478
t_ve_h323_h245_configuration_s 482
t_ve_h323_h450_2_configuration_s 484
t_ve_h323_h450_3_configuration_s 485
t_ve_h323_h450_4_configuration_s 486
t_ve_h323_h450_6_configuration_s 486
t_ve_h323_h450_7_configuration_s 487
t_ve_h323_h450_configuration_s 483
t_ve_h323_msg_center_id_s 468
t_ve_h323_name_id_s 474
t_ve_h323_q931_annex_e_configuration_s 481
t_ve_h323_q931_configuration_s 479
t_ve_h323_ras_configuration_s 477
t_ve_h323_system_configuration_s 476
t_ve_h323_user_address_s 467
t_ve_h323_user_channel_selection_s 472
t_ve_h323_user_registration_s 467
t_ve_init_config_s 254
t_ve_ip_default_attributes_s 261
t_ve_ip_default_configuration_s 260
t_ve_ip_settings_default_configuration_s 261
t_ve_last_stream_configuration_s 257
t_ve_line_ecan_16ms_ecan_configuration_s .. 265

t_ve_line_ecan_configuration_s 264
t_ve_line_ecan_configuration_u 264
t_ve_media_attrib_mapping_s 305
t_ve_media_cn_decoder_algorithm_configuration_s 324
t_ve_media_cn_encoder_algorithm_configuration_s 324
t_ve_media_definition_s 315
t_ve_media_description_s 302
t_ve_media_eg711_decoder_algorithm_configuration_s 338
t_ve_media_eg711_encoder_algorithm_configuration_s 337
t_ve_media_field_s 303
t_ve_media_g711_configuration_s 308
t_ve_media_g711_vad_decoder_algorithm_configuration_s 317
t_ve_media_g711_vad_encoder_algorithm_configuration_s 317
t_ve_media_g723_1_decoder_algorithm_configuration_s 319
t_ve_media_g723_1_encoder_algorithm_configuration_s 319
t_ve_media_g726_configuration_s 309
t_ve_media_g726_decoder_algorithm_configuration_s 320
t_ve_media_g726_encoder_algorithm_configuration_s 320
t_ve_media_g729a_configuration_s 308, 309
t_ve_media_g729a_decoder_algorithm_configuration_s 318
t_ve_media_g729a_encoder_algorithm_configuration_s 318
t_ve_media_gsm_amr_configuration_s 310
t_ve_media_gsm_amr_decoder_algorithm_configuration_s 323
t_ve_media_gsm_amr_encoder_algorithm_configuration_s 321
t_ve_media_h263_configuration_s 313
t_ve_media_ilbc_configuration_s 312
t_ve_media_ilbc_decoder_algorithm_configuration_s 340
t_ve_media_ilbc_encoder_algorithm_configuration_s 339
t_ve_media_ip_configuration_s 304
t_ve_media_mpeg4_configuration_s 314
t_ve_media_rtp_attributes 306
t_ve_media_rtp_configuration_u 307
t_ve_media_te_algorithm_configuration_s 325
t_ve_media_video_buffer_verifier_params_s 333
t_ve_media_video_compression_params_s 335
t_ve_media_video_cyclic_intra_refresh_s 331
t_ve_media_video_decoder_h263_algorithm_configuration_s 326
t_ve_media_video_decoder_mpeg4_algorithm_configuration_s 336
t_ve_media_video_encoder_h263_algorithm_configuration_s 325
t_ve_media_video_encoder_mpeg4_algorithm_configuration_s 327
t_ve_media_video_encoding_mode_rate_s 341
t_ve_media_video_frame_type_selection_s 336
t_ve_media_video_intra_control_s 334
t_ve_media_video_mpeg4_stream_container_s 342
t_ve_media_video_pid_rate_control_s 332
t_ve_media_video_pixel_aspect_ratio_s 329
t_ve_media_video_stream_type_info_s 330
t_ve_media_video_user_data_s 328
t_ve_nat_channel_addresses_s 492
t_ve_nat_configuration_s 496
t_ve_nat_default_configuration_s 491
t_ve_nat_init_s 494
t_ve_nat_media_addresses_s 493
t_ve_nat_sip_addresses_s 497
t_ve_nat_stun_init_s 495
t_ve_nat_turn_init_s 495
t_ve_net_default_configuration_s 499
t_ve_net_lookup_ipv4_address_s 501
t_ve_net_lookup_naptr_enum_s 503
t_ve_net_lookup_naptr_s 421, 502
t_ve_net_lookup_query_s 500
t_ve_net_lookup_record_ipv4_address_s 420
t_ve_net_lookup_record_s 419
t_ve_net_lookup_record_server_s 420
t_ve_net_lookup_server_s 501
t_ve_network_connection_s 498
t_ve_network_control_u 342
t_ve_ntd_algorithm_default_configuration_s 274
t_ve_ntg_algorithm_default_configuration_s 275
t_ve_null_post_proc_default_configuration_s 281
t_ve_null_pre_proc_default_configuration_s 276
t_ve_post_proc_default_configuration_s 280
t_ve_pre_proc_default_configuration_s 275
t_ve_remote_user_uri_s 466
t_ve_run_vericall_edge_config_s 258
t_ve_sip_authentication_data_s 450
t_ve_sip_call_id_s 390
t_ve_sip_contact_info_s 441
t_ve_sip_content_data_s 450
t_ve_sip_date_s 447
t_ve_sip_default_configuration _s 284
t_ve_sip_dialog_data_s 459
t_ve_sip_dialog_direction_s 462
t_ve_sip_dialog_duration_s 462
t_ve_sip_dialog_id_s 460
t_ve_sip_dialog_s 460
t_ve_sip_dialog_state_s 461

t_ve_sip_mw_data_s 457
t_ve_sip_mw_msg_summary_data_s 458
t_ve_sip_presence_data_s 452
t_ve_sip_presence_data_u 452
t_ve_sip_presence_xml_pidf_contact_s 456
t_ve_sip_presence_xml_pidf_note_s 456
t_ve_sip_presence_xml_pidf_presence_s 453
t_ve_sip_presence_xml_pidf_status_s 455
t_ve_sip_presence_xml_pidf_tuple_s 454
t_ve_sip_privacy_identity_s 464
t_ve_sip_privacy_s 465
t_ve_sip_registration_server_s 442
t_ve_sip_routing_data_s 445
t_ve_sip_routing_s 445
t_ve_sip_rx_session_response_content_u 449
t_ve_sip_rx_session_response_data_s 449
t_ve_sip_session_handle_s 463
t_ve_sip_stack_feature_support_s 443
t_ve_sip_stack_liberalisation_s 443
t_ve_sip_tx_session_response_content_u 448
t_ve_sip_tx_session_response_data_s 448
t_ve_sip_user_registration_s 440
t_ve_sip_user_uri_s 440
t_ve_stat_av_sync_s 433
t_ve_stat_channel_av_sync_s 432
t_ve_stat_channel_rtcp_peer_s 438
t_ve_stat_channel_rtcp_s 435
t_ve_stat_channel_video_decoder_s 434
t_ve_stat_channel_video_encoder_s 433
t_ve_stat_rtcp_s 436
t_ve_stat_system_calls_s 432
t_ve_stat_video_decoder_s 435
t_ve_stat_video_encoder_s 434, 439
t_ve_statistic_u 431
t_ve_stream_configuration_s 298
t_ve_system_logging_s 256
t_ve_system_s 253
t_ve_tone_definition_s 488
t_ve_tone_subgroup_s 489
t_ve_tone_subtone_s 490
t_ve_tx_lip_sync_default_configuration_s 287
t_ve_user_channel_association_s 301
t_ve_user_channel_details_s 300
t_ve_user_channel_selection_s 301
t_ve_user_handle_s 299
t_ve_user_msg_alarm_s 413
t_ve_user_msg_decode_media_change_event_s 423
t_ve_user_msg_detect_tone_1100_s 406
t_ve_user_msg_detect_tone_2100_s 405
t_ve_user_msg_detect_tone_cp_s 405
t_ve_user_msg_detect_tone_dtmf_s 404
t_ve_user_msg_encode_image_event_s 408
t_ve_user_msg_h323_call_forwarded_s 399
t_ve_user_msg_h323_call_hold_response_s 401
t_ve_user_msg_h323_call_request_s 397
t_ve_user_msg_h323_call_response_s 397
t_ve_user_msg_h323_call_started_s 395
t_ve_user_msg_h323_call_stopped_s 396
t_ve_user_msg_h323_call_updated_s 396
t_ve_user_msg_h323_forward_response_s 399
t_ve_user_msg_h323_message_waiting_s 402
t_ve_user_msg_h323_mwi_response_s 401
t_ve_user_msg_h323_registration_response_s 394
t_ve_user_msg_h323_request_failed_s 403
t_ve_user_msg_h323_transfer_response_s 398
t_ve_user_msg_h323_user_input_received_s 400
t_ve_user_msg_h323_video_feature_request_s 404
t_ve_user_msg_module_state_transition_event_s 412
t_ve_user_msg_nat_discover_response_s 360
t_ve_user_msg_nat_keep_alive_event_s 362
t_ve_user_msg_nat_request_failed_s 361
t_ve_user_msg_nat_user_addresses_response_s 360
t_ve_user_msg_net_lookup_response_s 418
t_ve_user_msg_net_request_failed_s 421
t_ve_user_msg_network_connection_event_s. 417
t_ve_user_msg_play_prompt_event_s 407
t_ve_user_msg_received_rfc2833_digit_s 406
t_ve_user_msg_record_prompt_event_s 407
t_ve_user_msg_s 356
t_ve_user_msg_shutdown_complete_s 417
t_ve_user_msg_sip_dialog_notify_rx_s 392
t_ve_user_msg_sip_dialog_subscribe_response_rx_s 393
t_ve_user_msg_sip_dialog_subscribe_rx_s 391
t_ve_user_msg_sip_info_response_rx_s 380
t_ve_user_msg_sip_info_rx_s 379
t_ve_user_msg_sip_invite_response_rx_s 367
t_ve_user_msg_sip_invite_rx_s 365
t_ve_user_msg_sip_message_response_rx_s 376
t_ve_user_msg_sip_message_rx_s 375
t_ve_user_msg_sip_mw_notify_response_rx_s 388
t_ve_user_msg_sip_mw_notify_rx_s 387
t_ve_user_msg_sip_operation_failed_s 383
t_ve_user_msg_sip_options_response_rx_s 378
t_ve_user_msg_sip_options_rx_s 377
t_ve_user_msg_sip_prack_rx_s 389
t_ve_user_msg_sip_presence_notify_rx_s 385
t_ve_user_msg_sip_presence_subscribe_response_rx_s 386
t_ve_user_msg_sip_presence_subscribe_rx_s. 384
t_ve_user_msg_sip_refer_notify_rx_s 374
t_ve_user_msg_sip_refer_response_rx_s 373
t_ve_user_msg_sip_refer_rx_s 372

t_ve_user_msg_sip_register_response_rx_s....371
t_ve_user_msg_sip_reinvite_response_rx_s ...370
t_ve_user_msg_sip_reinvite_rx_s 366
t_ve_user_msg_sip_service_response_rx_s 382
t_ve_user_msg_sip_service_rx_s 381
t_ve_user_msg_sip_session_started_s.....363
t_ve_user_msg_sip_session_stopped_s.....364
t_ve_user_msg_sip_session_updated_s.....364
t_ve_user_msg_sip_update_response_rx_s.....369
t_ve_user_msg_sip_update_rx_s.....368
t_ve_user_msg_stream_event_s.....411
t_ve_user_msg_undecodeable_stream_s.....422
t_ve_user_msg_video_encode_path_config_err_s 409
t_ve_version_s.....255
t_ve_video_control_data_end_point_selector_s 351
t_ve_video_control_image_encode_s 350
t_ve_video_control_lip_sync_adjustment_s....350
t_ve_video_control_move_and_resize_s.....349
t_ve_video_control_mute_s 349
t_ve_video_control_reset_post_proc_params_s 354
t_ve_video_control_reset_pre_proc_params_s352
t_ve_video_control_s 347
t_ve_video_control_scale_s 348
t_ve_video_post_proc_configuration_s.....282
t_ve_video_pre_proc_default_configuration_s277
t_ve_video_tear_protect_default_configuration_s 288
System API
Functions.....8
VE_ClearStatistics 37
VE_Commit 23
VE_GetNextEvent.....33
VE_GetStatistics 35
VE_GetSystemAttribute.....15
VE_GetVersion 34
VE_Initialize 11
VE_MemoryResourceQuery 10
VE_RegisterApp 9
VE_RemoveSystemAttribute 21
VE_RunVeriCallEdge 25
VE_RunVeriCallEdgeWithConfig 27
VE_SetSystemAttribute.....12
VE_Shutdown 31
VE_UpdateChannelAttribute.....19
VE_UpdateSystemAttribute 17

T

Terminology 669
Type Definitions 668

U

User API
Functions 40
VE_AddUser 41
VE_RemoveUser.....45
VE_UpdateUser.....43
VE_UpdateUserAttribute 47

V

VeriCall Edge
Architecture 5
Control Interface.....6
Data Interface 6
Introduction 5
Packet Interface 6